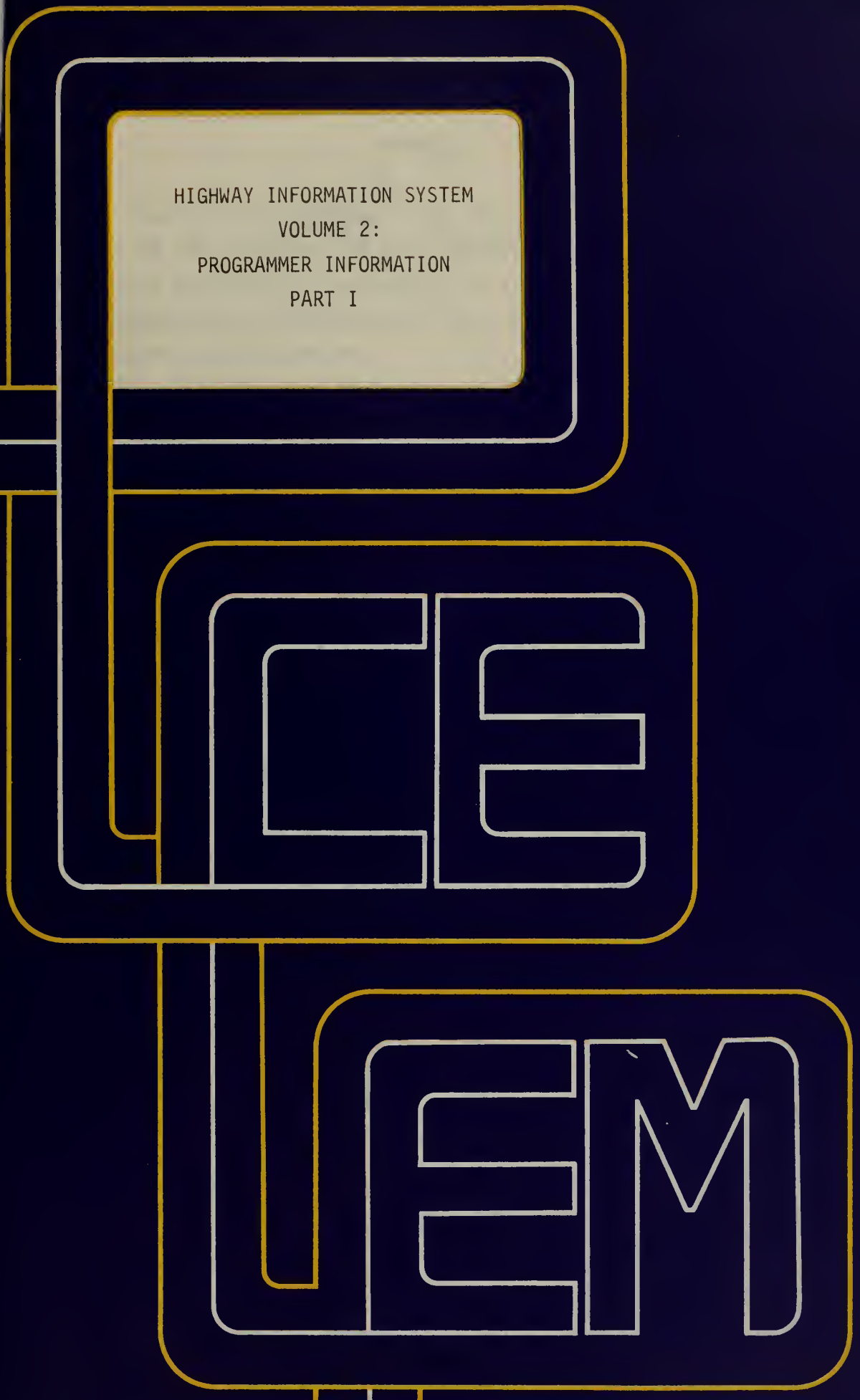
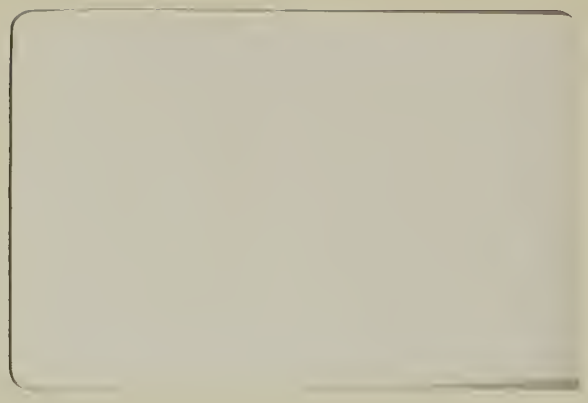
HIGHWAY INFORMATION SYSTEM
VOLUME 2:
PROGRAMMER INFORMATION
PART I



Department of Civil Engineering & Engineering Mechanics Montana State University Bozeman



HIGHWAY INFORMATION SYSTEM
VOLUME 2:
PROGRAMMER INFORMATION
PART I

Prepared for the

STATE OF MONTANA
DEPARTMENT OF HIGHWAYS
PLANNING AND RESEARCH BUREAU

In cooperation with the

U.S. DEPARTMENT OF TRANSPORTATION
FEDERAL HIGHWAY ADMINISTRATION

Prepared by

Glen L. Martin, Larry J. Coats, and Michael J. Meldahl

DEPARTMENT OF CIVIL ENGINEERING AND ENGINEERING MECHANICS
MONTANA STATE UNIVERSITY
Bozeman, Montana 59715

June, 1972

FOREWORD

This report is a documentation of the highway data bank research undertaken by the Department of Civil Engineering and Engineering Mechanics, Montana State University. The research was sponsored by the Montana Department of Highways in cooperation with the U.S. Department of Transportation, Federal Highway Administration.

Conceptually, the CE & EM Department was responsible for developing an information retrieval system for rapid access to highway data. Specifically, the responsibility was to produce the Roadlog, Traffic by Sections, Accident by Sections and Sufficiency by Sections reports as a direct application of the system. In addition, preliminary investigation of the feasiblity of a geometrics file and a preliminary investigation of the storage and retrieval of visual images was included in the project objectives.

In light of the foregoing, it is desirable to present the report in two volumes: Highway Information System Volume 1: User Information, and Highway Information System Volume 2: Programmer Information. Volume 1 deals with the use of the system, including information on data coding and on the execution of programs within the system. Volume 2 deals with the detailed operation of the system, providing information on the modification of programs existing within the system as well as on the addition of programs to the system. Volume 1 is a prerequisite publication to Volume 2.

In developing the system, the CE & EM Department has had the privilege of using an IBM OS 360/40 computer located at the Data Processing Bureau of the Montana Department of Highways in Helena. PL/I has been used as the programming language for nearly all of the HIS routines because of its versatility in input-output (I/O) and interchangeability of files. BAL (assembler) has been used for several routines because of its increased capabilities and efficiency over other languages.

The project could never have progressed to its current state were it not for the continual encouragement from and the patient, sustained assistance of both the Planning and Research Bureau and the Data Processing Bureau of the Montana Department of Highways, and of the Montana State Highway Patrol.

The project conclusion was also hastened by the significant effort of other project personnel: Francis C. F. Yu, Leroy R. Zook, Philip A. House,

Alfred C. Scheer, Paul W. Burkhart, Robert C. Smith, Harry E. Hughes, Ronald E. Billstein, Daniel D. Urbach and Donald R. Reichmuth. Their assistance has been invaluable.

HIGHWAY INFORMATION SYSTEM
VOLUME 2: PROGRAMMER INFORMATION

TABLE OF CONTENTS

Table of Contents (Continued)

Table of Contents (Continued)

Table of Contents (Continued)

LIST OF FIGURES

LIST OF TABLES

CHAPTER 2-I
INTRODUCTION

HIS consists of a large number of file-maintenance and summary-producing programs integrated into a user-oriented system. To aid the user in the use of the programs, a supervisor has been included in the system.

The supervisor is necessary for two reasons. First, the large number of programs, each executed separately, would cause a large amount of OS JCL to be required. Second, the supervisor aids in the communication between separate routines.

The user must communicate his needs to the supervisor by means of HIS command cards. These commands are written in a largely free-format user-oriented language, and are difficult to handle internally. To bridge this problem, a "command decoder" routine is provided which transforms the commands into a fixed-format record, called an "instruction." The supervisor reads each of the instructions, and executes them by loading the proper routine into core for execution.

HIS is designed to run in conjunction with the IBM System/360 Operating System (OS). Figure 2-I-1 shows the relationship between the HIS components and OS.

As can be seen from Figure 2-I-1, user input falls into three categories: 1) OS Job Control Language (JCL), 2) commands to the HIS supervisor, and 3) input data to HIS routines.

OS JCL is required in order to instruct the Operating System to execute the HIS supervisor. HIS commands are required to define the operations to be performed, and to execute the proper HIS routines. Input data is required by some of the HIS routines in order to update HIS data files.

The following signs and conventions, consistent with those used by IBM, have been adopted throughout these manuals:

1) Uppercase letters and punctuation marks (except for brackets and braces) must be coded.

2) Lowercase letters and terms represent information that must be supplied by the user.

3) Information contained within brackets [ ] represents an option that may be included or omitted, depending upon the requirements of the user.

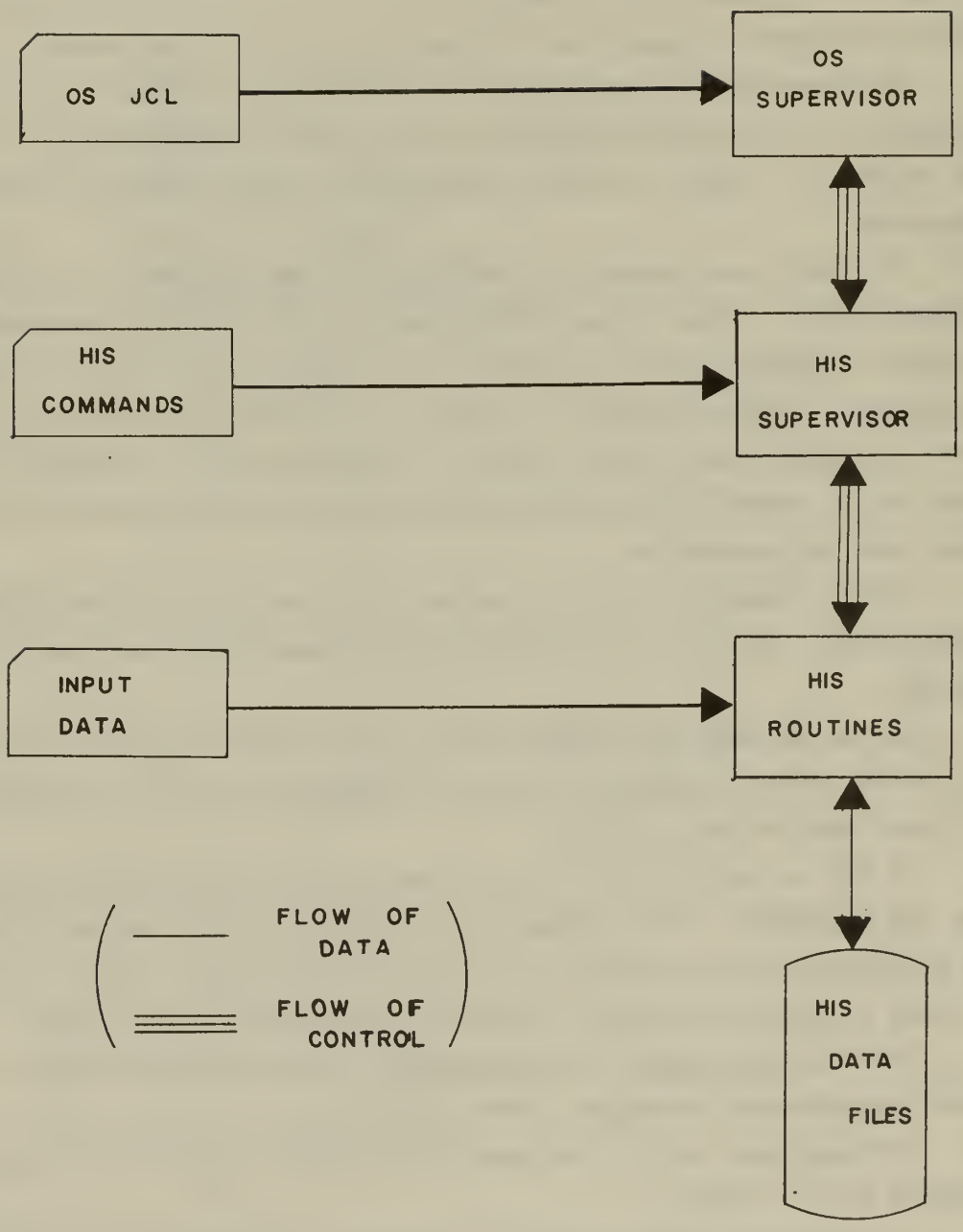


Figure 2-I-1. HIS organization.

4) Options contained within braces { } represent alternatives, one (and only one) of which must be chosen.

HIS Commands

HIS command cards are identified by a colon (:) in column 1. All commands must contain this identification. Immediately following the colon is coded the name of the HIS program to be executed.

Most routines allow one or more options to be selected by the user. These options are selected by means of parameters coded on the command following the program name. Each parameter consists of a keyword and an option, separated by an equal sign (e.g., FILE=ROADLOG). The first parameter is separated by a comma from the program name. Additional parameters are separated by commas. The last parameter must be followed by at least one blank.

Continuation cards -- When a command is too large to be contained on a single card, it may be continued on another card by placing a comma after a complete parameter, leaving the remainder of the card blank. The continuation card must contain a colon in column 1, followed by one or more blanks. Examples of HIS commands are:

```
:LIST-&-SUM,REPORT=ROADLOG,DATA=PRIM,PAGE-NUMBER=60
:SURF-TYPE,REPORT=ROADLOG,DATA=SEC,SUMMARY=RTE-NO,
:          MILEAGE=ALL,TABLE-NUMBER=6
:UPDATE,FILE=TRAFFIC,FUNCTION=INSERT,DDNAME=INDD
```

Instructions

Because the command format is largely free-form, it is rather unweildy to process. The commands are thus translated into a fixed-form record for internal handling. This internal format is referred to as an instruction.

The first four characters of the instruction give the name of a load module located in the step library which is to be executed. A portion of the name is computed directly from the program name specified on the command.

The remainder of the name may be filled in from parameters coded on the command.

The remaining characters of the instruction contain, in simplified form, the additional information included on the command. Each parameter is asigned a specific column, or columns, in the instruction as shown in Table 2-I-I.

For example, a number of programs are available under HIS for updating data files. These programs are invoked by specifying the program name UPDATE on the command. Three parameters are required on UPDATE commands: FILE, FUNCTION, and DDNAME. The file parameter indicates the name of the file to be updated (e.g., ROADLOG or TRAFFIC). The function parameter indicates whether records are to be inserted, deleted, or rewritten. The DDNAME parameter gives the name of a DD statement used to enter the update data. Three separate programs are available for each file; one performing each of the tasks: insertion, deletion, or revision. Hence, the FUNCTION parameter is used, along with the FILE parameter, to define the load module name. The internal name of the UPDATE program is PD. If FILE=ROADLOG and FUNCTION= INSERT are coded, an R appears in column 3, and an I appears in column 4. Hence, the load module name for this routine is PDRI. Similarly, when FILE=TRAFFIC,FUNCTION=DELETE is specified, the load module name is PDTD. The DDNAME parameter is not used to complete the load module name, but is examined by the update routine invoked when the instruction is executed to determine the location of its data.

Formatting of Output

A number of the HIS routines produce summaries which will be reproduced for printing directly from the computer printout. For this reason, a number of options are available to the user in the formatting of output. These formatting options affect all three of the HIS components (supervisor, command decoder, and file-maintenance and summary-producing programs).

Page size option -- The user must be able to vary the number of lines printed on each page. He does this by specifying PAGE-SIZE=nn in the command, where nn is the number of lines per page. The command decoder must recognize the PAGE-SIZE keyword, and place the number nn in columns 7 and 8 of the

TABLE 2-I-I

INSTRUCTION FORMAT

Parameter Name	Instruction Column(s)	Remarks
FILE	3	See Note 1 below.
REPORT	3	See Note 2 below.
SUMMARY	4	See Note 3 below.
FHSUMMARY	4	See Note 4 below.
FUNCTION	4	See Note 5 below.
PHASE	4	See Note 6 below.
MILEAGE	5	See Note 7 below.
LIST	5	See Note 8 below.
PAGE-SIZE	7-8	See Note 9 below.
TOP-MARGIN	11-12	See Note 9 below.
PAGE-NUMBER	15-19	See Note 9 below.
TABLE-NUMBER	21-22	See Note 9 below.
PAGE-EJECT	23	See Note 10 below.
DDNAME	24-31	See Note 11 below.
INDD	24-31	See Note 11 below.
START-DATE	24-31	See Note 11 below.
OUTDD	32-39	See Note 11 below.
END-DATE	32-39	See Note 11 below.
LOCATION	40-57	See Note 11 below.
STARTKEY	40-55	See Note 11 below.
ENDKEY	56-71	See Note 11 below.
ODD-PAGE-POSITION	73-75	See Note 9 below.
EVEN-PAGE-POSITION	76-78	See Note 9 below.
FORMAT	7-8 and 73-78	See Note 12 below.
DATA	6 and 40-71	See Note 13 below.
Miscellaneous Uses	remaining columns	See Note 14 below.

Notes: 1. The FILE parameter is used to complete the load module name. The allowable options and their corresponding internal codes are:

ROADLOG	"R"
TRAFFIC	"T"
TRUMILE	"M"
SUFFICIENCY	"S"
ACCIDENT	"A"

2. The REPORT parameter is used to complete the load module name. The allowable options and their corresponding internal codes are:

ROADLOG	"R"
TRAFFIC	"T"
SUFFICIENCY	"S"
ACCIDENT	"A"
SMTABLES	"Q"

TABLE 2-I-I (continued)

3. The SUMMARY parameter is used to complete the SURF-TYPE load module name. The allowable options and their corresponding internal codes are:

RTE-NO	"1"
PROJ-#	"2"
COUNTY	"3"
CITIES	"4"
YR-BLT	"5"
SUR-WD	"6"
YR-IMP	"7"

4. The FHSUMMARY parameter is used to complete the FORHWY-SUMMARY load module name. The allowable options and their corresponding internal codes are:

LOCATION	"L"
SURF-TYPE	"S"

5. The FUNCTION parameter is used to complete the load module name of UPDATE. The allowable options and their corresponding internal codes are:

DELETE	"D"
INSERT	"I"
REWRITE	"R"
NEW-KEY	"N"

6. The PHASE parameter is used to complete the load module names of CREATE-SUFFSUB and CREATE-ACCSUB. The allowable options and their corresponding internal codes are:

ACCIDENT	"A"
ROADLOG	"R"
TRAFFIC	"T"
SECTIONS	"S"
SUFFICIENCY	"S"
COMPUTATIONS	"C"

7. The MILEAGE parameter is used by SURF-TYPE to determine whether urban mileage or all mileage is to be processed. The allowable options and their corresponding internal codes are:

URBAN	"U"
ALL	"A"

8. The LIST parameter is used by COPY and CREATE to determine whether a "dump" listing is required. The allowable options and their internal codes are:

TABLE 2-I-I (continued)

YES	"Y"
NO	"N"

9. The PAGE-SIZE, TOP-MARGIN, PAGE-NUMBER, TABLE-NUMBER, ODD-PAGE-POSITION, and EVEN-PAGE-POSITION are numeric values coded for HIS formatting options. The numbers coded on the command are placed, in character format, into the proper columns of the instruction.
10. PAGE-EJECT=SUPPRESS is the only valid form of the PAGE-EJECT parameter. When this parameter is coded, an "S" is placed in instruction column 23.
11. These parameters provide the capability of linking a character string from the command to the HIS routines. The option coded in the parameter is copied directly into the appropriate columns of the instruction.
12. The FORMAT parameter is essentially a "macro" parameter. FORMAT=REDUCE is identical to the sequence PAGE-SIZE=60, EVEN-PAGE-POSITION=1, ODD-PAGE-POSITION=120. FORMAT=NOREDUCE is identical to the sequence PAGE-SIZE=46, EVEN-PAGE-POSITION=1, ODD-PAGE-POSITION=98.
13. The DATA parameter provides a versatile method of entering beginning and ending keys into the instruction. The decoder calculates the actual keys. Column 6 is given a value for the following options:

INT	"I"
PRIM	"P"
SEC	"S"
INT+PRIM	"C"
ILOOP	"L"
ALL	"A"

    All other options result in a blank in column 6. Columns 40-55 are filled with a starting key. Columns 56-71 are filled in with an ending key.
14. Columns 9-10 are used during the execution of a HIS routine to count the number of lines printed on a page. Column 72 is used during the execution of a HIS routine to indicate the number of page headings to be printed on each page. Columns 1-2 are the first portion of the load module name, and are filled in from the program name coded on the command. The remaining columns (13-14, 20, and 79-100) are presently unused.

instruction. If the PAGE-SIZE keyword is not coded on a command, columns 7 and 8 of the instruction will be left blank. The supervisor must check each instruction processed to see whether columns 7 and 8 contain a value. If no value is present, a default value is padded in. The default value at the beginning of system execution is 60 lines per page, but this value may be altered by the user for the duration of a run. Each of the HIS programs must check this field when printing, and print the correct number of lines on each page.

Page position -- The page position field of the instruction (columns 9 and 10) cannot be filled in by the user. The decoder will always leave these columns blank. The page position field is used to keep track of the present position of the line printer in order to detect when a new page is to be printed. The supervisor normally fills this field in with the same value as the page size, indicating to the called program that a new page is to be started. (When PAGE-EJECT=SUPPRESS is coded, this convention is not followed. See PAGE-EJECT option, below). The called program must always return the present page position in this field when control is returned to the supervisor.

Page numbering option -- The user must be able to number pages as they are printed. Several summaries may be printed serially by separate programs, and the pages numbered consecutively. Hence, each program has to return the present page number value to the supervisor when returning control. The decoder must recognize three forms of the page number option. The first form is PAGE-NUMBER=nnnn (four digits maximum). The number coded (preceded by blanks if shorter than four digits) is placed in columns 15 through 18 of the instruction, and column 19 is left blank. The second form is PAGE-NUMBER=$+nnnn, indicating that the value nnnn is to be added to the current page number to allow the insertion of maps or figures into a report. Again, the number nnnn is placed in columns 15 through 18 of the instruction. A dollar sign character ($) is placed in column 19. The final form, PAGE-NUMBER=STOP, is used to end page numbering. The decoder places a character "X" in column 15, leaving columns 16 through 19 blank. If no PAGE-NUMBER parameter is present on the command, the instruction will contain blanks in columns 15 through 19. The supervisor must keep track of page numbering. A

field of blanks is initially established, indicating that page numbering has not yet been initiated. After page numbering has started, this field contains the page numbers returned from called programs. If the page number field of an instruction contains blanks, this field is copied into the instruction. Hence, if the field is blanks, page numbering will not be initiated. If the field is not blanks, the next page number to be used is placed into the instruction. If PAGE-NUMBER=n was specified, the supervisor need do nothing because page number position of the instruction is already complete. If PAGE-NUMBER=$+n was specified, the supervisor must add the number coded to the value in the saved page number field. The result is filled into columns 15 through 18 of the instruction. If PAGE-NUMBER=STOP is specified, the supervisor must replace the "X" in column 15 with a blank, to indicate the absence of page numbering. The called program has the responsibility of printing the page numbers. If the page number field contains blanks, the called program does not print page numbers. If the page number field contains a value, the first new page started by the program contains this page number. The called program should then increment this value to indicate the next page number. Hence, when control is returned, this field contains the value of the next page number to be printed. It will be noted that the forms PAGE-NUMBER=$+nnnn and PAGE-NUMBER=STOP are processed entirely by the supervisor, and do not need to be checked for by the called program.

Page number positioning options -- Depending upon the method of printing and binding, the location at which page numbers will be printed on the page must be adjustable. Some of the reports printed require that odd page numbers be printed in a different location from even page numbers. The user indicates the position of page numbers by coding ODD-PAGE-POSITION=nnn and EVEN-PAGE-POSITION=nnn. The decoder must recognize the keywords when present, and fill the odd value into columns 73-75, and the even value into columns 76-78. If not coded on the command, the instruction fields will be blank. The supervisor must check these columns, and, if blank, fill in a default value. The default values for both fields are initially 120, but are adjustable by the user. The called program must consult these fields whenever a page number is to be printed in order to place the page number in the proper location. The decoder must also recognize the keyword FORMAT, with options NOREDUCE and REDUCE. FORMAT=REDUCE is equivalent to coding the three param-

eters PAGE-SIZE=60,ODD-PAGE-POSITION=120,EVEN-PAGE-POSITION=1. FORMAT=NOREDUCE is equivalent to coding the three parameters PAGE-SIZE=46, ODD-PAGE-POSITION=98,EVEN-PAGE-POSITION=1. The FORMAT option is decoded as if the three parameters were present; hence, the supervisor and called programs do not need to check for this option.

Table numbering options -- Many tables in the Federal Aid Roadlog report have a table number printed at the top. The user may have this identification generated by coding TABLE-NUMBER=nn on a command. Each page (if the summary generates more than one page) must have the table number printed. Table numbering continues from one command to the next, much like page numbering. However, each page produced by a single program contains the same identification. The table number is incremented before the next program begins to execute. TABLE-NUMBER=STOP can be coded to suppress table numbering. The decoder places the number nn in columns 21-22 of the instruction. If TABLE-NUMBER=STOP is coded, column 21 is set to "X." The supervisor must keep track of the current table number. If the table number parameter is not coded on a command, the supervisor must supply the present table number, if table numbering is in effect. If TABLE-NUMBER=STOP is coded, the supervisor must note this, and blank out the table number field. The called program must check each time it begins a new page whether a table number is to be printed. After the program has completed execution, it must increment the table number, and return the value to the supervisor.

Top margin option -- The user may cause an additional top margin to be printed at the top of each page. This is done by specifying TOP-MARGIN=nn on a command. The decoder places the value coded in columns 11 and 12. The supervisor need not examine this field, as the top margin option is never carried from one command to another. The called program must examine this field, and print nn blank lines at the top of each page.

PAGE-EJECT=SUPPRESS option -- This option is used when two small summaries are desired on the same page. (Normally, the supervisor sets the page-position field equal to the page-size field to force the output on a new page). The called programs must return the page position of its last line

of output for the PAGE-EJECT option to work. The decoder places the character "S" in column 23 of the instruction. The supervisor then copies the page-position returned from the previous program into the page-position field. The called program must examine column 23 if page numbering is in effect, as the page number should not be printed until a new page starts.

Setting default values -- When a number of programs will utilize the same values of page-size or other parameters, it is easier to set a default value for a system execution rather than coding the values on each command. This is done by specifying SYS-PARAM as the program name, and coding any of the parameters PAGE-SIZE, ODD-PAGE-POSITION, or EVEN-PAGE-POSITION. TABLE-NUMBER and PAGE-NUMBER may also be coded to start or terminate numbering of pages and tables. For example, if the user supplies the command:

```
:SYS-PARAM,PAGE-SIZE=40,EVEN-PAGE-POSITION=5,PAGE-NUMBER=3
```

the page-size default value is set to 40 and the even-page-position default value set to 5. The odd-page-position default is not altered. Page numbering (if not specified otherwise on the next command during the same system execution) will commence at number 3 when output is generated by a subsequent program. The status of table numbering is not altered. The decoder must place an "X" in column 1 (and blanks in columns 2-4) to indicate SYS-PARAM. The supervisor will not call in a program when this name is present, as it must handle default settings itself. The code is thus internal to the supervisor.

Program Descriptions

This section presents descriptions of the HIS supervisor and command decoder. The load modules of each of these programs are located in cataloged library HIS.LOADLIB. The load module names are given with the program description.

Supervisor --

Load Module Name . . . . . SUPER

Language . . . . . BAL

DD Statements Utilized . . . . INSTRCT -- instructions
STEPLIB -- location of load modules for execution

When invoked, the first act of the supervisor is to load and execute the command decoder (entry point name DECODE). The decoder reads the user's commands, decodes them into instructions, and places them in a temporary disk file, defined by DD statement INSTRCT. The record length of this file is 100 characters. After the decoder has finished its task, the supervisor again receives control. It must read each instruction, perform any processing required to formatting options, set up linkage to the instruction for the called program, and load and execute the appropriate program. (If the load module name specified is "X," no program is loaded, as the program SYS-PARAM was specified.) The requirements for setting up formatting options has already been described. The first four characters of the instruction comprise the program name. Because the LOAD macro used to load the program into storage requires an eight-byte field, these characters are moved into a double-word ("ENTRY") prior to issuing the LOAD macro. After executing the program, the supervisor must retain the returned values of the page number, page position, and table number fields. When the end-of-file condition is raised on INSTRCT, HIS execution terminates. Linkage of the instruction to the called program follows the System/360 OS convention for passing a parameter to a main program. Register 1 points to a parameter list, having one entry. The high-order bit is 1 to indicate the last entry. The remainder of the full-word entry points to the instruction (preceded by a half-word control field, indicating the length as being 100 characters). This linkage is depicted in Figure 2-I-2. Following is the program listing for the supervisor:

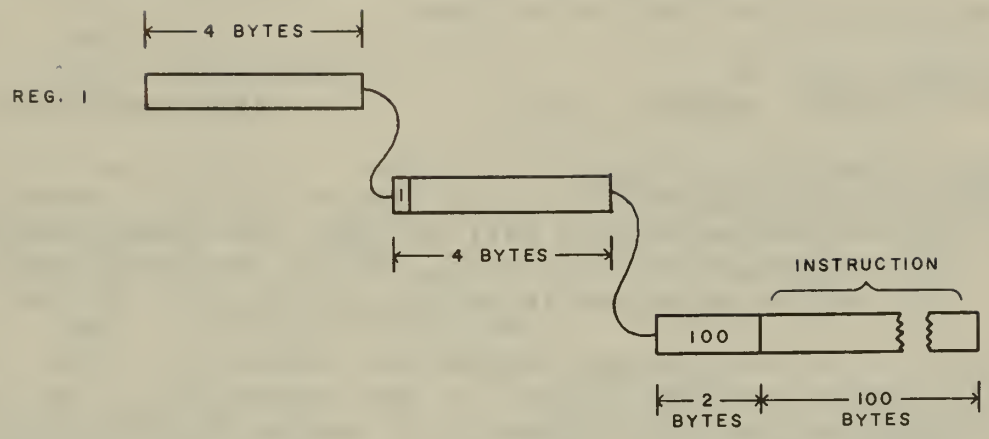


Figure 2-I-2. Linkage of instructions to called programs.

```
 1:* HIS SUPERVISOR
 2:SUPER    START
 3:         PRINT NOGEN
 4:BEGIN    EQU   *
 5:         PRIME
 6:*
 7:* LOAD AND EXECUTE DECODER ROUTINE
 8:*
 9:         LR    2,1                          SAVE CONTENTS OF R1
10:         LOAD  EP=DECODE                    RETRIEVE LOAD MODULE "DECODE"
11:         LR    1,2                          RESTORE R1
12:         LR    15,0                         ADDR OF DECODER ENTRY POINT
13:         BALR  14,15                        TRANSFER CONTROL TO DECODER
14:        DELETE EP=DECODE                    FREE CORE USED BY DECODER
15:         OPEN  (INSTRCT)                    FILE CONTAINS DECODED COMMANDS
16:*
17:* MAIN EXECUTION LOOP BEGINS HERE
18:*
19:READINST EQU   *
20:         GET   INSTRCT,INSTR                READ ONE INSTRUCTION
21:*
22:* PAGE NUMBERING OPTION.
23:* USER MAY SPECIFY:
24:*    1.  PAGE-NUMBER=STOP ('X' IN COL. 15, BLANKS IN COL. 16-19)
25:*    2.  PAGE-NUMBER=N (N IN COL. 15-18, BLANK IN COL. 19)
26:*    3.  PAGE-NUMBER=$+N (N IN COL. 15-18, '$' IN COL. 19)
27:*
28:* CASE 1:  NO PAGE-NUMBER PARAMETER PRESENT.
29:* "PAGENMBR" CONTAINS BLANKS IF PAGE NUMBERING IS NOT IN EFFECT.
30:* "PAGENMBR" CONTAINS CURRENT PAGE NUMBER IF PAGE NUMBERING IS IN
31:*    EFFECT.
32:*
33:         CLC   INSTR+14(4),=C'    '         IS PARAMETER PRESENT?
34:         BNE   PAGE05                       BR IF PRESENT
35:         MVC   INSTR+14(4),PAGENMBR         BLANKS OR PAGE NUMBER FROM
36:         B     ENDPAGE                         PREVIOUS PROGRAM
37:*
38:* CASE 2:  PAGE-NUMBER=STOP.  REPLACE 'X' WITH BLANK.
39:*
40:PAGE05   EQU   *
41:         CLI   INSTR+14,C'X'                PAGE-NUMBER=STOP?
42:         BNE   PAGE10                       BRANCH IF NO
43:         MVI   INSTR+15,C' '                BLANK OUT PAGE-NUMBER FIELD
44:         B     ENDPAGE
45:*
46:* CASE 3.  PAGE-NUMBER=N.  NO CHANGE NEEDS TO BE MADE.
47:*
48:PAGE10   EQU   *
49:         CLI   INSTR+18,C'$'                PAGE-NUMBER=$+N?
50:         BNE   ENDPAGE                      BRANCH IF PAGE-NUMBER=N FORM
51:*
52:* CASE 4.  PAGE-NUMBER=$+N.  ADD CONTENTS OF PAGENMBR TO
53:*    COL. 15-18 (BOTH ARE IN CHARACTER FORMAT)
54:*
55:         MVN   ZON1,PAGENMBR                CONVERT TO ZONED DECIMAL
56:         MVN   ZON2,INSTR+14                CONVERT TO ZONED DECIMAL
57:         PACK  PACK1,ZON1                   CONVERT TO PACKED DECIMAL
58:         PACK  PACK2,ZON2                   CONVERT TO PACKED DECIMAL
```

```
* HIS SUPERVISOR

 59:          AP     PACK2,PACK1              PERFORM ADDITION IN PACK2
 60:          UNPK   ZON2,PACK2               CONVERT TO ZONED DECIMAL
 61:          MVN    INSTR+14(4),ZON2         RETURN TO INSTR (CHAR FORMAT)
 62:ENDPAGE   EQU    *
 63:*
 64:* TABLE NUMBERING OPTION.
 65:* USER CODED TABLE-NUMBER=N OR TABLE-NUMBER=STOP
 66:*
 67:* CASE 1.  TABLE-NUMBER=STOP.  BLANK OUT TABLE-NUMBER FIELD.
 68:*
 69:          CLI    INSTR+20,C'X'
 70:          BNE    TABL05                   B IF NOT TABLE-NUMBER=STOP
 71:          MVI    INSTR+20,C' '            BLANK OUT FIELD
 72:          B      ENDTABL
 73:*
 74:* CASE 2.  TABLE-NUMBER=N.  NO CHANGE REQUIRED.
 75:*
 76:TABL05    EQU    *
 77:          CLC    INSTR+20(2),=C'  '
 78:          BNE    ENDTABL                  BRANCH IF TABLE-NUMBER=N
 79:*
 80:* CASE 3.  NO PARAMETER CODED.
 81:* "TABLNMBR" CONTAINS BLANKS IF TABLE NUMBERING NOT IN EFFECT.
 82:* "TABLNMBR" CONTAINS ONE PLUS THE PREVIOUS TABLE NUMBER IF TABLE
 83:*    NUMBERING IS IN EFFECT.
 84:*
 85:          MVC    INSTR+20(2),TABLNMBR
 86:ENDTABL   EQU    *
 87:*
 88:* PAGE-SIZE OPTION.
 89:* USER CODED PAGE-SIZE=N.  VALUE CODED IN IS COL. 7-8 OF INSTR.
 90:* IF NOT CODED, "PAGESIZE" CONTAINS DEFAULT VALUE TO BE USED.
 91:*
 92:          CLC    INSTR+6(2),=C'  '
 93:          BNE    *+10                     B IF PARAMETER IS PRESENT
 94:          MVC    INSTR+6(2),PAGESIZE      FILL IN DEFAULT VALUE
 95:*
 96:* PAGE-EJECT=SUPPRESS OPTION.
 97:* IF NOT PRESENT, SET PAGE-POSITION FIELD EQUAL TO PAGE-SIZE FIELD
 98:*    IN ORDER TO FORCE OUTPUT ON NEW PAGE.
 99:* IF PRESENT, SET PAGE-POSITION FIELD EQUAL TO VALUE RETURNED FROM
100:*    PREVIOUS PROGRAM.
101:*
102:          CLI    INSTR+22,C'S'            PAGE-EJECT=SUPPRESS?
103:          BNE    *+14                     BRANCH IF NOT PRESENT
104:          MVC    INSTR+8(2),POSITION      VALUE RETURNED FROM PREV PGM
105:          B      *+10
106:          MVC    INSTR+8(2),INSTR+6       USE PAGE-SIZE FIELD
107:*
108:* ODD-PAGE-POSITION OPTION.
109:* USER CODES ODD-PAGE-POSITION=N
110:* N IS DECODED INTO COL. 73-75
111:* IF NOT PRESENT, USE DEFAULT VALUE IN "ODDPOS"
112:*
113:          CLC    INSTR+72(3),=C'   '
114:          BNE    *+10                     BRANCH IF OPTION PRESENT
115:          MVC    INSTR+72(3),ODDPOS       DEFAULT VALUE
116:*
```

```
117:* EVEN-PAGE-POSITION OPTION.
118:* USER CODED EVEN-PAGE-POSITION=N
119:* N IS DECODED INTO COL 76-78
120:* IF NOT CODED, USE DEFAULT VALUE IN "EVENPOS"
121:*
122:         CLC   INSTR+75(3),=C'   '
123:         BNE   *+10                      BRANCH IF OPTION PRESENT
124:         MVC   INSTR+75(3),EVENPOS       USE DEFAULT VALUE
125:*
126:*
127:* AT THIS POINT, ALL FORMATTING OPTIONS HAVE BEEN PROCESSED.
128:* IF PROGRAM SPECIFIED WAS SYS-PARAM (INTERNAL NAME 'X'), DEFAULT
129:* VALUES ARE TO BE SET.
130:*
131:         CLC   INSTR(2),=C'X '           SYS-PARAM PROGRAM?
132:         BE    SYSPARAM                  BRANCH IF YES
133:*
134:* FIRST FOUR CHARACTERS OF INSTR GIVE THE NAME OF THE LOAD MODULE
135:*   TO BE LOADED AND EXECUTED.  IF THE LOAD MODULE DOES NOT EXIST
136:*   IN THE STEP OR SYSTEM LIBRARIES, THE JOB WILL ABEND WITH
137:*   SYSTEM COMPLETION CODE 806.
138:*
139:         MVC   ENTRY(4),INSTR
140:*
141:* PROGRAMS "SURF-TYPE" AND "SUMMARY-BY-ROUTES" REQUIRE
142:*   ADDITIONAL PROCESSING TO DETERMINE LOAD MODULE NAME.
143:*
144:         CLC   INSTR(2),=C'ND'           :SURF-TYPE?
145:         BE    SURFTYPE
146:         CLC   INSTR(3),=C'NCR'          :SUMMARY-BY-LOCATION?
147:         BE    SUMLOCN
148:*
149:* RETRIEVE LOAD MODULE
150:*
151:LOADIT   EQU   *
152:         LOAD  EPLOC=ENTRY               ENTRY ADDR RETURNED TO RO
153:*
154:* SET UP LINKAGE TO INSTRUCTION AND PASS CONTROL
155:*
156:         LA    1,INSTRPTR
157:         LR    15,0
158:         BALR  14,15
159:*
160:* FREE STORAGE AND SAVE RETURNED VALUES OF FORMATTING PARAMETERS
161:*
162:        DELETE EPLOC=ENTRY
163:         MVC   PAGENMBR,INSTR+14         SAVE PAGE NUMBER
164:         MVC   POSITION,INSTR+8          SAVE PAGE POSITION
165:         MVC   TABLNMBR,INSTR+20         SAVE TABLE NUMBER.
166:         B     READINST                  GET NEXT INSTRUCTION AND
167:*                                        EXECUTE IT.
168:* :SURF-TYPE -- SET UP LOAD MODULE NAME
169:*
170:SURFTYPE EQU   *
171:         MVI   ENTRY+3,C' '
172:         CLI   INSTR+3,C'1'              SUMMARY=RTE-NO?
173:         BNE   LOADIT                    BRANCH IF NO
174:         CLI   INSTR+5,C'L'              DATA=ILOOP?
```

```
* HIS SUPERVISOR

 175:          BE    LOADIT                    BRANCH IF YES
 176:          MVI   ENTRY+3,C'1'
 177:          B     LOADIT
 178:*
 179:* :SUMMARY-BY-LOCATION -- SET UP LOAD MODULE NAME
 180:*
 181:SUMLOCN   EQU   *
 182:          MVC   ENTRY+3(1),INSTR+5        DATA CODE FIELD
 183:          B     LOADIT
 184:*
 185:* :SYS-PARAM PROGRAM -- SET DEFAULT VALUES
 186:*
 187:SYSPARAM  EQU   *
 188:          MVC   PAGESIZE,INSTR+6          STORE PAGESIZE DEFAULT
 189:          MVC   ODDPOS,INSTR+72           ODD-PAGE-POSITION DEFAULT
 190:          MVC   EVENPOS,INSTR+75          EVEN-PAGE-POSITION DEFAULT
 191:          MVC   PAGENMBR,INSTR+14         SAVE PAGE NUMBER
 192:          MVC   TABLNMBR,INSTR+20         SAVE TABLE NUMBER
 193:          B     READINST                  GET NEXT INSTRUCTION
 194:*
 195:* COME HERE AFTER ALL INSTRUCTIONS HAVE BEEN EXECUTED.
 196:*
 197:ENDFILE   EQU   *
 198:          CLOSE (INSTRCT)
 199:          TERME
 200:*
 201:*
 202:* VARIABLE DECLARATIONS *
 203:*
 204:*
 205:PAGESIZE  DC    C'60'                     INITIAL DEFAULT VALUE
 206:ODDPOS    DC    C'120'                    INITIAL DEFAULT VALUE
 207:EVENPOS   DC    C'120'                    INITIAL DEFAULT VALUE
 208:POSITION  DC    C'99'                     INIT VALUE -- START NEW PAGE
 209:PAGENMBR  DC    C'    '                   INIT VALUE -- NO PAGE NUMBERS
 210:TABLNMBR  DC    C'  '                     INIT VALUE -- NO TABLE NUMBERS
 211:          CNOP  2,4
 212:PARAMTR   DC    H'100'                    LENGTH OF PARAMETER PASSED TO
 213:INSTR     DS    CL100                        CALLED PROGRAM IS 100 CHAR
 214:ENTRY     DC    CL8' '                    FOR LOAD MODULE NAMES
 215:          DS    0F                        FULLWORD BOUNDARY
 216:INSTRPTR  DC    X'80'
 217:          DC    AL3(PARAMTR)
 218:ZON1      DC    ZL4'0'                    FOR DATA CONVERSION
 219:ZON2      DC    ZL4'0'                    FOR DATA CONVERSION
 220:PACK1     DS    PL3                       FOR DATA CONVERSION
 221:PACK2     DS    PL3                       FOR DATA CONVERSION
 222:INSTRCT   DCB   DSORG=PS,MACRF=GM,DDNAME=INSTRCT,EODAD=ENDFILE
 223:          END   BEGIN
```

Command decoder --

Load Module Name . . . . . DECODE

Language . . . . . BAL

DD Statements Utilized . . . . . PRINTER -- printer output
PROGTBL -- table of program names
INSTRCT -- instructions
SYSIN -- user-supplied commands

The command decoder has the task of translating user-supplied commands into the more easily processed instruction format. Each command has the general format:

:pgmname,keyword=option,...

The program name may be up to 20 characters in length. The keyword may be any length, but only the first 10 characters are examined. The option fields are limited to 40 characters. Load module names are limited by OS to eight characters; the HIS instruction format limits HIS load module names to four characters. Hence, a conversion must be made from the longer "external" program name coded on the command to an "internal" name of four or fewer characters. A conversion table giving each of the allowable external names and the corresponding internal names is stored on disk (or other medium), and is defined by DD statement PROGTBL. Each of the records in the table is eighty bytes in length (to allow card input), containing an external name in columns 1 through 20, and an internal name in columns 21 through 24. The decoder begins its execution by reading the program table into a core array. As the table is read, it is printed for reference by the user when the table must be updated. After the program table is read, the decoder can begin decoding the commands. Each command is tested to ensure a colon as the first character. A character-by-character search is then performed to find the first blank or comma -- the first such character terminates the program name. The program name is moved into location "PGM," and padded with blanks if necessary to fill 20 characters. The name is then compared with each entry in the program table, and the proper internal name substituted into the instruction. If no match is found, the first four characters of the program name coded on the command are used as the internal name; this facility allows

the testing of programs not yet entered into the program table by coding the internal name on a command. For each parameter present on the command, the keyword is retrieved (in the same character-by-character method used to retrieve the program name), and placed into "KEYWORD." The option is retrieved and placed into "OPTION." The keyword is then searched against a table of keywords (lines 616-642 in the listing). If no match is found, an error message is printed. Otherwise, a branch table is constructed to branch to a routine to decode the parameter (lines 189-216). A blank following any parameter or program name terminates the command. The completed instruction is then written (lines 524-530) into the instruction file, and printed in the command listing. After all the commands have been decoded, the instruction and other files are closed, and return is made to the supervisor.

The program listing for the command decoder follows:

```
 1:* HIS COMMAND DECODER ROUTINE
 2:DECODER  START
 3:         PRINT NOGEN
 4:BEGIN    EQU   *
 5:         PRIME
 6:         BALR  11,0
 7:         USING *,11,12
 8:         LA    12,2048(11)
 9:         LA    12,2048(12)
10:         OPEN  (TABLE,,PRNT,(OUTPUT))   PROGRAM TABLE & PRINTER OUTPU
11:         PUT   PRNT,P1                  HEADING
12:         PUT   PRNT,P2                  BLANK LINE
13:* READ TABLE OF PROGRAM NAMES FOR CONVERSION FROM EXTERNAL TO
14:*   INTERNAL NAMES.
15:*
16:* FORMAT OF TABLE ENTRIES:
17:*    COL  1-20  EXTERNAL NAME (20 CHARS MAX LENGTH)
18:*    COL 21-24  INTERNAL NAME (4 CHARS MAX LENGTH)
19:*    COL 25-80  NOT PROCESSED -- MAY CONTAIN ANY CHARACTERS
20:         LA    3,EXTERNAL               ADDR OF EXTERNAL NAME ARRAY
21:         LA    4,INTERNAL               ADDR OF INTERNAL NAME ARRAY
22:         LA    5,0                      COUNTS NUMBER OF TABLE ENTRIE
23:         LA    6,50                     50 ENTRIES MAXIMUM ALLOWWED
24:PROGLOOP EQU   *
25:         GET   TABLE                    READ A TABLE ENTRY
26:         MVC   0(20,3),0(1)             MOVE EXT NAME TO EXT ARRAY
27:         MVC   0(4,4),20(1)             MOVE INT NAME TO INT ARRAY
28:         MVC   P4+5(80),0(1)            PRINT THE RECORD
29:         PUT   PRNT,P4
30:         LA    3,20(3)                  R3 = R3 + 20
31:         LA    4,4(4)                   R4 = R4 + 4
32:         LA    5,1(5)                   R5 = R5 + 1
33:         BCT   6,PROGLOOP
34:CLOSTABL EQU   *                        COME HERE AT END-OF-FILE
35:         CLOSE (TABLE)
36:         STH   5,PROGNO                 NUMBER OF ENTRIES PROCESSED
37:* OPEN FILES FOR COMMAND INPUT AND INSTRUCTION OUTPUT
38:         OPEN  (SYSIN,,INSTRCT,(OUTPUT))
39:*
40:* MAIN EXECUTION LOOP BEGINS HERE
41:*
42:READCARD EQU   *
43:         GET   SYSIN                    READ A COMMAND
44:         LR    3,1                      R3 = ADDR OF COMMAND
45:         NI    CHARCNTR+1,X'00'         RESET COUNTER
46:         MVC   P2+5(80),0(3)            PRINT THE COMMAND
47:         CLI   PAGECNTR+1,X'38'         PAGE FULL?
48:         BNH   PRNTLINE                 BRANCH IF NO
49:         PUT   PRNT,P1                  HEADING
50:         MVI   PAGECNTR+1,X'00'         RESET PAGE COUNTER
51:PRNTLINE EQU   *
52:         PUT   PRNT,P2
53:         LH    4,PAGECNTR               PAGECNTR = PAGECNTR + 2
54:         LA    4,2(4)
55:         STH   4,PAGECNTR
56:         BAL   14,GETCHR                GET FIRST CHARACTER
57:         CLI   CHAR,C'>'                COMMENT CARD?
58:         BE    READCARD
```

```
* HIS COMMAND DECODER ROUTINE

 59:          CLI   CHAR,C':'                 FIRST CHARACTER MUST BE COLON
 60:          BE    GETPGM
 61:NOCOLON   EQU   *
 62:          MVC   P3+5(20),*+10             PRINT ERROR MESSAGE
 63:          B     PRNTERR
 64:          DC    CL20'COLON MISSING'
 65:*
 66:* RETRIEVE PROGRAM NAME
 67:*
 68:GETPGM    EQU   *
 69:          MVI   PGM,C' '                  PGM = BLANKS
 70:          MVC   PGM+1(20),PGM
 71:          LA    4,21                      MAX PGM NAME LENGTH + 1
 72:          LA    5,PGM
 73:GETPGM05  EQU   *
 74:          BAL   14,GETCHR                 GET NEXT CHARACTER
 75:          CLI   CHAR,C' '                 BLANK/COMMA TERMINATES NAME
 76:          BE    HAVEPGM
 77:          CLI   CHAR,C','
 78:          BE    HAVEPGM
 79:          MVC   0(1,5),CHAR               BUILD NAME IN PGM
 80:          LA    5,1(5)                    INCREMENT R5
 81:          BCT   4,GETPGM05
 82:          MVC   P3+5(20),*+10             PRINT ERROR MESSAGE
 83:          B     PRNTERR
 84:          DC    CL20'PGM NAME TOO LONG'
 85:*
 86:* SEARCH EXTERNAL NAME ARRAY FOR PROGRAM NAME
 87:*
 88:HAVEPGM   EQU   *
 89:          LA    4,EXTERNAL                ADDR OF EXT NAME ARRAY
 90:          LA    5,INTERNAL                ADDR OF INT NAME ARRAY
 91:          LH    6,PROGNO                  NO. OF ENTRIES IN TABLE
 92:HAVEPGM1  EQU   *
 93:          CLC   PGM(20),0(4)              COMPARE NAME TO ARRAY ENTRY
 94:          BE    FOUNDPGM
 95:          LA    4,20(4)                   R4 = R4+20
 96:          LA    5,4(5)                    R5 = R5+4
 97:          BCT   6,HAVEPGM1
 98:* IF NAME IS NOT FOUND IN THE TABLE, USE THE FIRST FOUR CHARACTERS
 99:*   OF THE NAME CODED ON THE COMMAND FOR INTERNAL NAME
100:          MVC   INSTR(4),PGM
101:          B     FOUNDPGM+6
102:*
103:* PROGRAM NAME FOUND IN TABLE.  R5 POINTS TO INTERNAL NAME
104:*
105:FOUNDPGM  EQU   *
106:          MVC   INSTR(4),0(5)
107:          MVI   INSTR+4,C' '              FILL REST OF INSTR WITH BLANKS
108:          MVC   INSTR+5(95),INSTR+4
109:*
110:* LOOP FOR DECODING PARAMETERS.
111:* EACH PARAMETER IS WRITTEN AS KEYWORD=OPTION
112:*
113:PARMLOOP  EQU   *
114:          CLI   CHAR,C' '                 LAST PARM FOLLOWED BY BLANK
115:          BE    WRITINST
116:          MVI   KEYWORD,C' '              SET KEYWORD, OPTION TO BLANKS
```

```
* HIS COMMAND DECODER ROUTINE

117:          MVC   KEYWORD+1(50),KEYWORD
118:          BAL   14,GETCHR                  GET NEXT CHARACTER
119:          CLI   CHAR,C' '                  CONTINUATION CARD REQUIRED?
120:          BE    CONTINUE
121:* NOTE:   ONLY FIRST TEN CHARS OF KEYWORD ARE EXAMINED
122:GETKEY    EQU   *
123:          LA    4,KEYWORD
124:          LA    5,10                       MAX NUMBER CHARS TO BE READ
125:GETKEY05  EQU   *
126:          CLI   CHAR,C'='                  TERMINATES KEYWORD
127:          BE    HAVEKEY
128:          MVC   0(1,4),CHAR                STORE CHAR INTO KEYWORD
129:          LA    4,1(4)                     INCREMENT R4
130:          BAL   14,GETCHR                  GET NEXT CHARACTER
131:          LTR   15,15                      END OF CARD? - ERROR
132:          BNZ   KEYERR
133:          BCT   5,GETKEY05
134:* ONLY COME HERE IF KEYWORD IS LONGER THAN 10 CHARACTERS.  IGNORE
135:* THE REMAINING CHARACTERS.
136:GETKEY10  EQU   *
137:          CLI   CHAR,C'='
138:          BE    HAVEKEY
139:          BAL   14,GETCHR                  GET NEXT CHARACTER
140:          LTR   15,15                      END OF CARD? - ERROR
141:          BZ    GETKEY10
142:KEYERR    EQU   *
143:          MVC   P3+5(20),*+10              PRIN ERROR MESSAGE
144:          B     PRNTERR
145:          DC    CL20'INVALID KEYWORD'
146:*
147:* AFTER RETRIEVING KEYWORD, OBTAIN OPTION
148:*
149:HAVEKEY   EQU   *
150:          LA    4,OPTION
151:          LA    5,41                       MAX OPTION LENGTH + 1
152:GETOPT05  EQU   *
153:          BAL   14,GETCHR                  GET NEXT CHARACTER
154:          LTR   15,15                      TEST FOR END OF CARD
155:          BZ    GETOPT10                   BRANCH IF NOT END
156:          MVI   CHAR,C' '                  FLAG END OF COMMAND
157:          B     HAVEOPT
158:GETOPT10  EQU   *
159:          CLI   CHAR,C' '                  BLANK/COMMA TERMINATES OPTION
160:          BE    HAVEOPT
161:          CLI   CHAR,C','
162:          BE    HAVEOPT
163:          MVC   0(1,4),CHAR                BUILD OPTION
164:          LA    4,1(4)                     INCREMENT R4
165:          BCT   5,GETOPT05
166:OPTERR    EQU   *
167:          MVC   P3+5(20),*+10              PRINT ERROR MESSAGE
168:          B     PRNTERR
169:          DC    CL20'INVALID OPTION'
170:*
171:* AFTER OBTAINING KEYWORD AND OPTION, SEARCH FOR KEYWORD IN TABLE
172:* THE LAST ENTRY IN THE KEYWORD TABLE IS BLANKS
173:*
174:HAVEOPT   EQU   *
```

```
* HIS COMMAND DECODER ROUTINE

175:          LA    5,KEYS                    ADDR OF KEYWORD TABLE
176:          LA    6,0                       INIT COUNTER
177:SEARCH05  EQU   *
178:          CLC   0(10,5),=CL10' '          END OF TABLE -- NO MATCH FOUND
179:          BE    KEYERR
180:          CLC   0(10,5),KEYWORD           COMPARE KEYWORD TO TABLE ENTRY
181:          BE    FOUNDKEY
182:          LA    5,10(5)                   POINTS TO NEXT TABLE ENTRY
183:          LA    6,4(6)                    INCREMENT COUNTER
184:          B     SEARCH05
185:*
186:* COME HERE WHEN A MATCH HAS BEEN FOUND IN THE KEYWORD TABLE
187:* R6 CONTAINS 4 X THE NUMBER OF THE KEYWORD ENTRY
188:*
189:FOUNDKEY EQU   *
190:          B     *+4(6)
191:          B     FILE
192:          B     REPORT
193:          B     FUNCTION
194:          B     PHASE
195:          B     FHSUM
196:          B     SUMMARY
197:          B     MILEAGE
198:          B     LIST
199:          B     ACIDENTS
200:          B     DDNAME
201:          B     INDD
202:          B     STARTDTE
203:          B     OUTDD
204:          B     ENDDTE
205:          B     EJECT
206:          B     PAGESIZE
207:          B     TOPMRGN
208:          B     TABLNMBR
209:          B     ODDPOS
210:          B     EVENPOS
211:          B     PAGENMBR
212:          B     FORMAT
213:          B     LOCATION
214:          B     STARTKEY
215:          B     ENDKEY
216:          B     DATA
217:*
218:* FILE PARAMETER.  COL 3.
219:*
220:FILE      EQU   *
221:          CLC   OPTION(7),=C'TRUMILE'
222:          BNE   FILE05
223:          MVI   INSTR+2,C'M'
224:          B     PARMLOOP
225:FILE05    EQU   *
226:          MVC   INSTR+2(1),OPTION
227:          B     PARMLOOP
228:*
229:* REPORT PARAMETER.  COL 3.
230:*
231:REPORT    EQU   *
232:          CLC   OPTION(8),=C'SMTABLES'
```

```
* HIS COMMAND DECODER ROUTINE

233:          BNE   REPORT05
234:          MVI   INSTR+2,C'Q'
235:          B     PARMLOOP
236:REPORT05  EQU   *
237:          MVC   INSTR+2(1),OPTION
238:          B     PARMLOOP
239:*
240:* FUNCTION, PHASE, FHSUMMARY PARAMETERS -- COL 4
241:*
242:FUNCTION  EQU   *
243:FHSUM     EQU   *
244:PHASE     EQU   *
245:          MVC   INSTR+3(1),OPTION
246:          B     PARMLOOP
247:*
248:* SUMMARY PARAMETER -- COL 4
249:*
250:SUMMARY   EQU   *
251:          CLC   OPTION(6),=C'RTE-NO'
252:          BE    SUM1
253:          CLC   OPTION(6),=C'PROJ-#'
254:          BE    SUM2
255:          CLC   OPTION(6),=C'COUNTY'
256:          BE    SUM3
257:          CLC   OPTION(6),=C'CITIES'
258:          BE    SUM4
259:          CLC   OPTION(6),=C'YR-BLT'
260:          BE    SUM5
261:          CLC   OPTION(6),=C'SUR-WD'
262:          BE    SUM6
263:          CLC   OPTION(6),=C'YR-IMP'
264:          BE    SUM7
265:          MVC   P3+5(20),*+10
266:          B     PRNTERR                    PRINT ERROR MESSGE
267:          DC    CL20'INVALID SUMMARY PARM'
268:SUM1      EQU   *
269:          MVI   INSTR+3,C'1'
270:          B     PARMLOOP
271:SUM2      EQU   *
272:          MVI   INSTR+3,C'2'
273:          B     PARMLOOP
274:SUM3      EQU   *
275:          MVI   INSTR+3,C'3'
276:          B     PARMLOOP
277:SUM4      EQU   *
278:          MVI   INSTR+3,C'4'
279:          B     PARMLOOP
280:SUM5      EQU   *
281:          MVI   INSTR+3,C'5'
282:          B     PARMLOOP
283:SUM6      EQU   *
284:          MVI   INSTR+3,C'6'
285:          B     PARMLOOP
286:SUM7      EQU   *
287:          MVI   INSTR+3,C'7'
288:          B     PARMLOOP
289:*
290:* MILEAGE, LIST, ACCIDENTS PARAMETERS -- COL 5
```

```
* HIS COMMAND DECODER ROUTINE

291:*
292:MILEAGE  EQU   *
293:LIST     EQU   *
294:ACIDENTS EQU   *
295:         MVC   INSTR+4(1),OPTION
296:         B     PARMLOOP
297:*
298:* DDNAME, INDD, AND START-DATE PARAMETERS
299:* FIRST 8 CHARACTERS OF OPTION ARE PLACED INTO COL 24-31
300:*
301:DDNAME   EQU   *
302:INDD     EQU   *
303:STARTDTE EQU   *
304:         MVC   INSTR+23(8),OPTION
305:         B     PARMLOOP
306:*
307:* OUTDD AND END-DATE PARAMETERS
308:* MOVE FIRST 8 CHARACTERS OF OPTION INTO COL 32-39
309:*
310:OUTDD    EQU   *
311:ENDDTE   EQU   *
312:         MVC   INSTR+31(8),OPTION
313:         B     PARMLOOP
314:*
315:* PAGE-EJECT=SUPPRESS OPTION
316:*
317:EJECT    EQU   *
318:         MVC   INSTR+22(1),OPTION
319:         B     PARMLOOP
320:*
321:* 2-DIGIT NUMERIC OPTIONS
322:*
323:PAGESIZE EQU   *
324:         LA    4,INSTR+6                 COL 7-8
325:         B     NUMBER2D
326:TOPMRGN  EQU   *
327:         LA    4,INSTR+10                COL 11-12
328:         B     NUMBER2D
329:TABLNMBR EQU   *
330:         LA    4,INSTR+20                COL 21-22
331:         CLC   OPTION(5),=C'STOP '       TABLE-NUMBER=STOP FORM?
332:         BNE   NUMBER2D                  BRANCH IF NO
333:         MVI   INSTR+20,C'X'
334:         B     PARMLOOP
335:NUMBER2D EQU   *
336:         LA    5,OPTION
337:NMBR2D05 EQU   *
338:         MVC   0(1,4),1(4)               SHIFT LEFT
339:         MVC   1(1,4),0(5)               INSERT NEW DIGIT
340:         LA    5,1(5)                    INCREMENT R5
341:         CLI   0(5),C' '                 BLANK TERMINATES NUMBER
342:         BNE   NMBR2D05
343:         B     PARMLOOP
344:*
345:* THREE-DIGIT NUMERIC OPTIONS
346:*
347:ODDPOS   EQU   *
348:         LA    4,INSTR+72                COL 73-75
```

```
* HIS COMMAND DECODER ROUTINE

349:          B      NUMBER3D
350:EVENPOS   EQU    *
351:          LA     4,INSTR+75                COL 76-78
352:NUMBER3D  EQU    *
353:          LA     5,OPTION
354:NMBR3D05  EQU    *
355:          MVC    0(2,4),1(4)               SHIFT LEFT
356:          MVC    2(1,4),0(5)               INSERT NEW DIGIT
357:          LA     5,1(5)                    INCREMENT R5
358:          CLI    0(5),C' '                 BLANK TERMINATES NUMBER
359:          BNE    NMBR3D05
360:          B      PARMLOOP
361:*
362:* PAGE-NUMBER PARAMETER
363:*
364:PAGENMBR  EQU    *
365:          CLC    OPTION(5),=C'STOP '       PAGE-NUMBER=STOP FORM?
366:          BNE    PAGEN005                  BRANCH IF NO
367:          MVI    INSTR+14,C'X'
368:          B      PARMLOOP
369:PAGEN005  EQU    *
370:          LA     4,INSTR+14                COL 15-18
371:          CLC OPTION(2),=C'$+'             PAGE-NUMBER=$+N FORM?
372:          BNE    NUMBER4D                  BRANCH IF NOT $+N FORM
373:          MVI    INSTR+18,C'$'             $ IN COL 19
374:          LA     5,OPTION+2                INCREMENT PAST "$+"
375:          B      NMBR4D05
376:NUMBER4D  EQU    *
377:          LA     5,OPTION
378:NMBR4D05  EQU    *
379:          MVC    0(3,4),1(4)               SHIFT LEFT
380:          MVC    3(1,4),0(5)               INSERT NEW DIGIT
381:          LA     5,1(5)                    INCREMENT R5
382:          CLI    0(5),C' '                 BLANK TERMINATES NUMBER
383:          BNE    NMBR4D05
384:          B      PARMLOOP
385:*
386:* FORMAT PARAMETER
387:* FORMAT=REDUCE IS EQUIVALENT TO PAGE-SIZE=60, ODD-PAGE-POSITION=120,
388:*    AND EVEN-PAGE-POSITION=1
389:* FORMAT=NOREDUCE IS EQUIVALENT TO PAGE-SIZE=46, ODD-PAGE-POSITION=98,
390:*    AND EVEN-PAGE-POSITION=1
391:*
392:FORMAT    EQU    *
393:          CLI    OPTION,C'N'
394:          BE     NOREDUCE
395:          MVC    INSTR+6(2),=C'60'
396:          MVC    INSTR+72(6),=C'120001'
397:          B      PARMLOOP
398:NOREDUCE EQU    *
399:          MVC    INSTR+6(2),=C'46'
400:          MVC    INSTR+72(6),=C'098001'
401:          B      PARMLOOP
402:*
403:* LOCATION PARAMETER -- MOVE 18 CHARACTERS INTO COL 40-57
404:*
405:LOCATION  EQU    *
406:          MVC    INSTR+39(18),OPTION
```

```
* HIS COMMAND DECODER ROUTINE

407:          B      PARMLOOP
408:*
409:* STARTKEY PARAMETER -- MOVE 16 CHARACTERS INTO COL 40-55
410:*
411:STARTKEY  EQU    *
412:          MVC    INSTR+39(16),OPTION
413:          B      PARMLOOP
414:*
415:* ENDKEY PARAMETER -- MOVE 16 CHARACTERS INTO COL 56-71
416:*
417:ENDKEY    EQU    *
418:          MVC    INSTR+55(16),OPTION
419:          B      PARMLOOP
420:*
421:* CODE FOR DECODING DATA PARAMETER
422:* POSSIBLE FORMATS ARE:
423:*   DATA=INT  DATA=PRIM  DATA=SEC  DATA=INT+PRIM  DATA=ALL  DATA=ILOOP
424:*    DATA=INT=N-N  DATA=PRIM=N-N  DATA=SEC=N-N
425:*
426:* INFORMATION MUST BE TRANSFORMED INTO A STARTING AND AN ENDING KEY
427:*    INTO THE APPROPRIATE FILE
428:*
429:DATA       EQU    *
430:          CLC    OPTION(3),=C'ALL'
431:          BE     DATAALL
432:          CLC    OPTION(8),=C'INT+PRIM'
433:          BE     DATAIP
434:          CLC    OPTION(3),=C'INT'
435:          BE     DATAI
436:          CLC    OPTION(4),=C'PRIM'
437:          BE     DATAP
438:          CLC    OPTION(3),=C'SEC'
439:          BE     DATAS
440:          CLC    OPTION(5),=C'ILOOP'
441:          BE     DATALP
442:DATAERR   EQU    *
443:          MVC    P3+5(20),*+10
444:          B      PRNTERR
445:          DC     CL20'INVALID DATA PARM'
446:DATAALL   EQU    *
447:          MVC    INSTR+39(4),=C'I015'      FIRST INTERSTATE RECORD
448:          MVC    INSTR+55(4),=C'S999'
449:          MVI    INSTR+5,C'A'              FLAG "DATA=ALL" FORM
450:DATARQ    EQU    *
451:          CLI    INSTR+2,C'R'              FILE=ROADLOG
452:          BE     PARMLOOP                  BRANCH IF YES
453:          CLI    INSTR+2,C'Q'              REPORT=SMTABLES (ROADLOG FILE)
454:          BE     PARMLOOP                  BRANCH IF YES
455:          MVC    INSTR+43(9),=C'000+0.000'
456:          B      PARMLOOP
457:DATAIP    EQU    *
458:          MVC    INSTR+39(4),=C'I015'      FIRST INTERSTATE RECORD
459:          MVC    INSTR+55(4),=C'P999'      LARGER THAN LAST PRIM RECORD
460:          MVI    INSTR+5,C'C'              FLAG "DATA=INT+PRIM" FORM
461:          B      DATARQ
462:DATAI     EQU    *
463:          MVC    INSTR+39(4),=C'I015'      FIRST INTERSTATE RECORD
464:          MVC    INSTR+55(4),=C'I999'      LARGER THAN LAST INT RECORD
```

```
* HIS COMMAND DECODER ROUTINE

465:          CLI   OPTION+3,C'='           DATA=INT=N-N FORM
466:          BE    DATARTE                 FILL IN ROUTE NUMBERS
467:          MVI   INSTR+5,C'I'            FLAG "DATA=INT" FORM
468:          B     DATARQ
469:DATAP     EQU   *
470:          MVC   INSTR+39(4),=C'P001'    FIRST PRIMARY RECORD
471:          MVC   INSTR+55(4),=C'P999'    LARGER THAN LAST PRIM RECORD
472:          CLI   OPTION+4,C'='           DATA=PRIM=N-N FORM
473:          BE    DATARTE                 FILL IN ROUTE NUMBERS
474:          MVI   INSTR+5,C'P'            FLAG "DATA=PRIM" FORM
475:          B     DATARQ
476:DATAS     EQU   *
477:          MVC   INSTR+39(4),=C'S201'    FIRST SECONDARY RECORD
478:          MVC   INSTR+55(4),=C'S999'    LARGER THAN LAST SEC RECORD
479:          CLI   OPTION+3,C'='           DATA=SEC=N-N FORM
480:          BE    DATARTE                 FILL IN ROUTE NUMBERS
481:          MVI   INSTR+5,C'S'            FLAG "DATA=SEC" FORM
482:          B     DATARQ
483:DATARTE   EQU   *                       COL 6 REMAINS BLANK FOR THESE
484:*                                       FORMS.
485:          MVC   INSTR+40(3),=C'000'     ZERO OUT RTE NO
486:          LA    4,INSTR+40              ADDR OF ROUTE FIELD
487:          LA    5,OPTION+4              LOCATION OF NO. SPECIFIED
488:          CLI   INSTR+39,C'P'           DATA=PRIM=N-N -- EXTRA CHAR
489:          BNE   *+8
490:DATARTE2  EQU   *
491:          LA    5,1(5)                  INCREMENT R5
492:          CLI   0(5),C'-'               HYPHEN/BLANK ENDS ROUTE NO
493:          BE    DATARTE4
494:          CLI   0(5),C' '
495:          BE    DATARTE8
496:          MVC   0(2,4),1(4)             SHIFT ONE PLACE
497:          MVC   2(1,4),0(5)             INSERT NEW DIGIT
498:          B     DATARTE2
499:DATARTE4  EQU   *
500:          MVC   INSTR+56(3),=C'000'     ZERO OUT ENDING RTE NO
501:          MVC   INSTR+59(9),=C'999+9.999'
502:          LA    4,INSTR+56
503:DATARTE6  EQU   *
504:          LA    5,1(5)                  IGNORE HYPHEN
505:          CLI   0(5),C' '               BLANK TERMINATES RTE NO
506:          BE    DATARQ
507:          MVC   0(2,4),1(4)             SHIFT LEFT
508:          MVC   2(1,4),0(5)             INSERT NEW DIGIT
509:          B     DATARTE6
510:DATARTE8  EQU   *                       COME HERE WHEN NO SECOND ROUTE
511:          MVC   INSTR+56(3),INSTR+40       SPECIFIED.  USE STARTING
512:          MVC   INSTR+59(9),=C'999+9.999'  ROUTE NUMBER.
513:          B     DATARQ
514:DATALP    EQU   *
515:          MVC   INSTR+39(4),=C'P001'    FIRST PRIMARY RECORD
516:          MVC   INSTR+55(4),=C'P999'    LARGER THAN LAST PRIM ENTRY
517:          MVI   INSTR+5,C'L'            FLAG "DATA=ILOOP" FORM
518:          B     PARMLOOP
519:*
520:*
521:* WRITE THE INSTRUCTION TO OUTPUT FILE
522:*
```

```
* HIS COMMAND DECODER ROUTINE

523:*
524:WRITINST EQU   *
525:         PUT   INSTRCT,INSTR
526:         MVC   P5+10(100),INSTR          PRINT THE INSTRUCTION
527:         PUT   PRNT,P5
528:         LH    4,PAGECNTR                INCREMENT PAGE COUNTER
529:         LA    4,1(4)
530:         STH   4,PAGECNTR
531:         B     READCARD
532:*
533:*
534:* GETCHR SUBROUTINE
535:* THIS SUBROUTINE RETRIEVES ONE CHARACTER FROM THE COMMAND.
536:* THE CHARACTER IS PLACED IN LOCATION "CHAR".
537:* R3 POINTS TO CURRENT CHARACTER
538:* R14 IS THE LINK REGISTER
539:* "CHARCNTR" COUNTS NUMBER OF CHARACTERS PROCESSED.
540:* IF ALL 80 CHARACTERS HAVE BEEN PROCESSED, "CHAR" IS SET TO BLANK,
541:*    AND R15 IS SET TO RETURN CODE OF 4.
542:* OTHERWISE, R15 IS SET TO ZERO.
543:*
544:*
545:GETCHR   EQU   *
546:         CLI   CHARCNTR+1,X'4F'          END OF CARD?
547:         BNE   GETCHR05                  BRANCH IF NO
548:         MVI   CHAR,C' '
549:         LA    15,4                      ABNORMAL RETURN CODE
550:         BR    14
551:GETCHR05 EQU   *
552:         MVC   CHAR,0(3)                 NEXT CHARACTER
553:         CLI   CHAR,C'_'
554:         BNE   *+8
555:         MVI   CHAR,C'-'
556:         LA    3,1(3)                    INCREMENT POINTER
557:         LH    15,CHARCNTR               INCREMENT COUNTER
558:         LA    15,1(15)
559:         STH   15,CHARCNTR
560:         LA    15,0                      NORMAL RETURN CODE
561:         BR    14
562:*
563:*
564:* CODE FOR CONTINUATION CARDS
565:*
566:*
567:CONTINUE EQU   *
568:         GET   SYSIN
569:         LR    3,1                       SAVE ADDR OF CARD
570:         NI    CHARCNTR+1,X'00'          RESET COUNTER
571:         MVC   P2+5(80),0(3)             PRINT THE CARD
572:         MVI   P2,C' '
573:         PUT   PRNT,P2
574:         MVI   P2,C'0'
575:         LH    4,PAGECNTR                INCREMENT COUNTER
576:         LA    4,1(4)
577:         STH   4,PAGECNTR
578:         BAL   14,GETCHR                 GET FIRST CHARACTER
579:         CLI   CHAR,C':'
580:         BNE   NOCOLON
```

```
* HIS COMMAND DECODER ROUTINE

581:          LA    4,79                        BEGIN SEARCH FOR NON-BLANK
582:LO5       EQU   *                              CHARACTER
583:          BAL   14,GETCHR                   GET NEXT CHARACTER
584:          CLI   CHAR,C' '
585:          BNE   GETKEY
586:          BCT   4,LO5
587:          MVC   P3+5(20),*+10               PRINT ERROR MESSAGE
588:          B     PRNTERR
589:          DC    CL20'NULL CONTINUATION'
590:*
591:* PRINT ERROR MESSAGE
592:*
593:PRNTERR   EQU   *
594:          PUT   PRNT,P3
595:          LH    4,PAGECNTR                  INCREMENT COUNTER
596:          LA    4,1(4)
597:          STH   4,PAGECNTR
598:          B     READCARD
599:*
600:*
601:* COME HERE WHEN FINISHED (EOF ON SYSIN)
602:*
603:*
604:RETURN    EQU   *
605:          CLOSE (PRNT,,SYSIN,,INSTRCT)
606:          TERME
607:*
608:*
609:* VARIABLES
610:*
611:*
612:PAGECNTR  DC    H'60'
613:CHARCNTR  DS    H
614:KEYWORD   DS    CL10
615:OPTION    DS    CL41
616:KEYS      EQU   *
617:          DC    CL10'FILE'
618:          DC    CL10'REPORT'
619:          DC    CL10'FUNCTION'
620:          DC    CL10'PHASE'
621:          DC    CL10'FHSUMMARY'
622:          DC    CL10'SUMMARY'
623:          DC    CL10'MILEAGE'
624:          DC    CL10'LIST'
625:          DC    CL10'ACCIDENTS'
626:          DC    CL10'DDNAME'
627:          DC    CL10'INDD'
628:          DC    CL10'START-DATE'
629:          DC    CL10'OUTDD'
630:          DC    CL10'END-DATE'
631:          DC    CL10'PAGE-EJECT'
632:          DC    CL10'PAGESIZE'
633:          DC    CL10'TOP-MARGIN'
634:          DC    CL10'TABLE-NUMBER'
635:          DC    CL10'ODD-PAGE-POSITION'
636:          DC    CL10'EVEN-PAGE-POSITION'
637:          DC    CL10'PAGE-NUMBER'
638:          DC    CL10'FORMAT'
```

```
* HIS COMMAND DECODER ROUTINE

639:          DC    CL10'LOCATION'
640:          DC    CL10'STARTKEY'
641:          DC    CL10'ENDKEY'
642:          DC    CL10'DATA'
643:          DC    CL10' '
644:P1        DC    CL133'1  HIS COMMAND DECODER'
645:P2        DC    CL133'0'
646:P3        DC    CL133' ***'
647:P4        DC    CL133' '
648:P5        DC    CL133' '
649:PGM       DS    CL21
650:INSTR     DS    CL100
651:CHAR      DS    CL1
652:PROGNO    DS    H
653:EXTERNAL  DS    50CL20
654:INTERNAL  DS    50CL4
655:PRNT      DCB   DSORG=PS,MACRF=PM,DDNAME=PRINTER
656:TABLE     DCB   DSORG=PS,MACRF=GL,DDNAME=PROGTBL,EODAD=CLOSTABL
657:SYSIN     DCB   DSORG=PS,MACRF=GL,DDNAME=SYSIN,EODAD=RETURN
658:INSTRCT   DCB   DSORG=PS,MACRF=PM,DDNAME=INSTRCT,BLKSIZE=800,LRECL=100, X
659:                RECFM=FB
660:          END   BEGIN
```

Printer subroutine

Object Module Name	PRINTX1
Language	PL/I
DD Statements Utilized	PRINTER and SYSPRINT
Entry Points	INIT EXIT PRINTX PRINTXA ASTER

As can be seen from the description given of formatting options available to the users of HIS, a great burden is placed on each program in the system in order to follow all the required conventions. The printer subroutine has been provided to ease this load when writing PL/I programs to run under HIS. The printer subroutine relieves the programmer of the responsibilities for the PAGE-SIZE, PAGE-NUMBER, TABLE-NUMBER, TOP-MARGIN, PAGE-NUMBER-POSITION, and PAGE-EJECT options. In addition, it keeps track of the printer location, advances to a new page when necessary, and can print page headings for the programmer. When a program receives control from the HIS supervisor, the 100-byte instruction is passed to it in the normal System/360 conventions. Hence, the PL/I programmer obtains the instruction by coding:

```
pgm-name:  PROCEDURE (PARM) OPTIONS (MAIN);
     DECLARE PARM CHAR(100);
```

Upon receiving control, the PL/I program must invoke the printer routine at entry point INIT, passing the instruction. The actions performed by INIT are: 1) open the printer file, 2) move the first 80 bytes of the instruction into external variable INSTR, and 3) set the external area HEADING to blanks.

In order to print a line of output, the line is set up in external variable PRINTER, and the printer subroutine is invoked at entry point PRINTX. A number is passed to indicate printer spacing. The numbers 0 through 8 indicate the number of lines to be skipped before printing (hence, 0 causes the line to be printed over the line just printed, 1 causes single-spacing, etc.). The number 9 is used to cause a skip to the next page. The subroutine is instructed to print page headings by placing a number indicating

the number of lines to be printed as headings in column 72 of the instruction, and placing the headings in external array HEADING. An example program is:

```
/* TEST PROGRAM SHOWING PRINT ROUTINE UTILIZATION */
TESTIT:  PROCEDURE (PARM) OPTIONS (MAIN);
   DECLARE PARM CHAR(100), INSTR CHAR(80) EXT, PRINTX ENTRY (PIC'Z'),
      (PRINTER,HEADING(9)) CHAR(132) EXT,
      #_HDGS PIC'Z' DEF INSTR POS(72);
   CALL INIT (PARM);
   #_HDGS = 2;
   HEADING(1) = '          PAGE HEADING';
   PRINTER = '     ONE LINE OF OUTPUT';
   CALL PRINTX (1);
   PRINTER = '    MORE OUTPUT';
   CALL PRINTX (6);
   CALL EXIT (PARM);
   END TESTIT;
```

The programmer has indicated to the print routine that two lines of headings will be required. A value has been placed into HEADING(1), the first heading line. No value has been placed into HEADING(2); this line will contain blanks because entry point INIT fills the heading array with blanks. When the print routine is first invoked at entry point PRINTX, a control number of "1" has been passed, indicating single spacing. However, the supervisor has passed values in the page-size and page-position fields of the instruction indicating that a new page must be started. Hence, the printer will be advanced to a new page, and page numbers, table numbers, and top margins printed as needed. The headings specified in the program are then printed. The line of output in PRINTER is then printed, spacing one line. When the second CALL PRINTX statement is encountered, the print routine (assuming the page size is larger than 6) will space down six lines, and print the second line. The output will then appear as:

```
PAGE HEADING
ONE LINE OF OUTPUT
  :
  :
(5 blank lines)
  :
  :
MORE OUTPUT
```

The statement "CALL EXIT (PARM);" is required after all print calls have been made. This entry point: 1) closes the printer file, and 2) copies INSTR into PARM to return values to the HIS supervisor.

The print routine has two additional entry points. Entry point ASTER prints a line containing 132 asterisks; this is handy to set off error messages that are otherwise easily unnoticed. Entry point PRINTXA can be used to ensure that a summary can fit on one page. For example, assume that an unknown amount of output has already been printed, and a thirteen-line summary is to be printed. If there are thirteen lines left on the current page, the summary is to be printed on the same page as the previous output. If there are less than thirteen lines left, a new page is to be started. The statement "CALL PRINTXA (1,13)" will insure that a new page is started if there are less than 13 lines remaining on the current page. The first number passed has the same meaning as when calling PRINTX, and the line to be printed is placed in PRINTER. An example program is:

```
DECLARE PARM CHAR(100), INSTR CHAR(80) EXT, #_HDGS PIC'Z' DEF
        INSTR POS(72), (PRINTER,HEADING(9)) CHAR(132) EXT,
        PRINTX ENTRY (PIC'Z'), PRINTXA ENTRY (PIC'Z',PIC'ZZ');
   CALL INIT (PARM);
        :
        :
   PRINTER = '   FIRST LINE OF SUMMARY';
   CALL PRINTXA (1,13);
   PRINTER = '   NEXT LINE';
   CALL PRINTX (1);
        :
        :
   CALL EXIT (PARM);
```

Following is the program listing for the printer subroutine:

```
/* SUBROUTINE HANDLING LINE-PRINTING FOR HIS ROUTINES */

 1: /* SUBROUTINE HANDLING LINE-PRINTING FOR HIS ROUTINES */
 2: PRINTX:  PROCEDURE (F);

 3: /* CONTROL VARIABLES IN HIS INSTRUCTION */
 4: DECLARE
 5:    INSTR CHAR(80) EXT,
 6:    PAGESIZE PIC'ZZ' DEF INSTR POS(7),
 7:    PAGE_POSITION PIC'ZZ' DEF INSTR POS(9),
 8:    TOP_MARGIN PIC'ZZ' DEF INSTR POS(11),
 9:    PAGE_NUMBER PIC'ZZZZ' DEF INSTR POS(15),
10:    TABLE_NUMBER PIC'ZZ' DEF INSTR POS(21),
11:    EJECT_CODE CHAR(1) DEF INSTR POS(23),
12:    ODDPOS PIC'ZZZ' DEF INSTR POS(73),
13:    EVENPOS PIC'ZZZ' DEF INSTR POS(76),
14:    #_HDGS PIC'Z' DEF INSTR POS(72);

15: /* INTERNAL VARIABLES */
16: DECLARE
17:    #_LINES PIC'ZZ',
18:    ASTS CHAR(132) STATIC INIT ((132)'*'),
19:    BLANKS CHAR(132) STATIC INIT (' '),
20:    F PIC'Z',
21:    FRMT(0:3) CHAR(1) STATIC INIT ('+',' ','0','-'),
22:    F0 DEC FIXED (1,0) STATIC,
23:    P CHAR(133) STATIC,
24:    P1 CHAR(1) DEF P,
25:    P2 CHAR(132) DEF P POS(2),
26:    PARM CHAR(100);

27: /* EXTERNAL VARIABLES */
28: DECLARE
29:    HEADING(9) CHAR(132) EXT,
30:    PRINTER CHAR(132) EXT;


31: START:
32:    F0 = F;

33: START1:

34:    /* NEW PAGE REQUIRED? */
35:    IF PAGE_POSITION+F0>PAGESIZE | F0>8 THEN DO;
36:       P1 = '1';
37:       P2 = ' ';
38:       PAGE_POSITION = 0;

39:       /* PAGE NUMBERING? */
40:       IF PAGE_NUMBER¬=0 THEN DO;
41:          J1 = PAGE_NUMBER;
42:          J2 = J1/2;  J2 = J2*2;
43:       IF J1=J2
44:          THEN SUBSTR(P2,EVENPOS,4) = PAGE_NUMBER;
45:          ELSE SUBSTR(P2,ODDPOS,4) = PAGE_NUMBER;
46:          WRITE FILE (PRNT) FROM (P);
47:          PAGE_NUMBER = PAGE_NUMBER + 1;
48:          P1 = ' ';
49:          PAGE_POSITION = 1;
50:          END;
```

```
/* SUBROUTINE HANDLING LINE-PRINTING FOR HIS ROUTINES */

51:       /* TABLE NUMBERING? */
52:       IF TABLE_NUMBER¬=0 THEN DO;
53:          P2 = SUBSTR(BLANKS,1,54) || 'TABLE NUMBER ' || TABLE_NUMBER;
54:          WRITE FILE (PRNT) FROM (P);
55:          P1 = '0';
56:          PAGE_POSITION = PAGE_POSITION + 2;
57:          END;

58:       /* TOP MARGIN? */
59:       J1 = TOP_MARGIN;
60:       P2 = ' ';
61:       DO WHILE (J1>0);
62:          IF P1='1' THEN WRITE FILE (PRNT) FROM (P);
63:          IF J1>3 THEN DO;
64:             P1 = '-';
65:             WRITE FILE (PRNT) FROM (P);
66:             J1 = J1 - 3;
67:             END;
68:          ELSE DO;
69:             P1 = FRMT(J1);
70:             J1 = 0;
71:             END;
72:          END;
73:       PAGE_POSITION = PAGE_POSITION + TOP_MARGIN;

74:       /* PAGE HEADINGS? */
75:       IF #_HDGS¬=0 THEN DO J1=1 TO #_HDGS;
76:          P2 = HEADING(J1);
77:          WRITE FILE (PRNT) FROM (P);
78:          P1 = ' ';
79:          END;
80:       PAGE_POSITION = PAGE_POSITION + #_HDGS;
81:       IF F0>8 THEN F0 = 1;
82:       EJECT_CODE = ' ';
83:       IF P1='1' THEN DO;
84:          P2 = ' ';
85:          WRITE FILE (PRNT) FROM (P);
86:          END;
87:       END;
88:    PAGE_POSITION = PAGE_POSITION + F0;

89:    /* CHECK FOR PAGE_EJECT=SUPPRESS */
90:    IF EJECT_CODE='S' THEN DO;
91:       IF PAGE_POSITION+#_HDGS+10>PAGESIZE THEN DO;
92:          PAGE_POSITION = PAGESIZE;
93:          GOTO START;
94:          END;
95:       P = '-';
96:       DO J1=1 TO 2;
97:          WRITE FILE (PRNT) FROM (P);
98:          END;
99:       IF #_HDGS¬=0 THEN DO J1=1 TO #_HDGS;
100:         P = ' ' || HEADING(J1);
101:         WRITE FILE (PRNT) FROM (P);
102:         END;
103:      PAGE_POSITION = PAGE_POSITION + #_HDGS + 6;
104:      EJECT_CODE = ' ';
```

```
/* SUBROUTINE HANDLING LINE-PRINTING FOR HIS ROUTINES */

105:        END;
106:     DO WHILE (FO>3);
107:        P = '-';
108:        WRITE FILE (PRNT) FROM (P);
109:        FO = FO - 3;
110:        END;
111:     P = FRMT(FO) || PRINTER;
112:     WRITE FILE (PRNT) FROM (P);
113:     RETURN;


114: /* ENTRY PRINTXA FOR TESTING NUMBER OF LINES LEFT ON PAGE */
115: PRINTXA:  ENTRY (F,#_LINES);
116:     IF PAGE_POSITION+#_LINES>PAGESIZE THEN DO;
117:        PAGE_POSITION = PAGESIZE;
118:        FO = 1;
119:        END;
120:     ELSE FO = F;
121:     GOTO START1;


122: /* ENTRY INIT FOR HIS INITIALIZATION */
123: INIT:  ENTRY (PARM);
124:     OPEN FILE (PRNT) OUTPUT RECORD TITLE ('PRINTER');
125:     INSTR = PARM;
126:     HEADING = ' ';
127:     #_HDGS = 0;
128:     RETURN;


129: /* ENTRY POINT EXIT CLOSES PRINT FILE */
130: EXIT:  ENTRY (PARM);
131:     CLOSE FILE (PRNT);
132:     IF TABLE_NUMBER¬=0 THEN TABLE_NUMBER = TABLE_NUMBER + 1;
133:     PARM = INSTR;
134:     RETURN;


135: /* ENTRY POINT ASTER PRINTS LINE OF ASTERISKS */
136: ASTER:  ENTRY;
137:     P = ' ' || ASTS;
138:     WRITE FILE (PRNT) FROM (P);
139:     RETURN;

140: END PRINTX;
```

HIS1 Cataloged Procedure

HIS1 is a procedure cataloged for utilizing programs operating under HIS. It contains DD statements STEPLIB, SYSPRINT, PRINTER, INSTRCT, and PROGTBL:

```
// PROC XLIB='HIS.LOADTST'
//HIS EXEC PGM=SUPER
//STEPLIB DD DISP=SHR,DSNAME=&XLIB
//        DD DISP=SHR,DSNAME=HIS.LOADLIB
//PRINTER DD SYSOUT=A,DCB=(BLKSIZE=1300,LRECL=133,RECFM=FBA)
//SYSPRINT DD SYSOUT=A
//INSTRCT DD UNIT=SYSDA,SPACE=(TRK,(1,1))
//PROGTBL DD DISP=SHR,DSNAME=HIS.TABLES(PGMTBL)
```

The symbolic parameter XLIB may be used to respecify the data set name of a library containing executable load modules to be specified on commands.

The procedure contains all DD statements, other than SYSIN (user commands), used by the supervisor, command decoder, and printer subroutine. All other DD statements utilized by programs executed in a run must be supplied with the run.

Writing Programs Under HIS

Programs that do not require an instruction -- If a program is to be written that does not require an instruction passed to it (i.e., the formatting options will not be used, and options from the command do not need to be processed), the program is simply written (in any language), and cataloged into a library with a name of four or less characters. To execute the program, simply supply a command having a colon in column 1, and the program name in columns 2-5. In the following simple PL/I program that reads cards and prints them, the DD statement INPUT is used to read cards:

```
PGM:  PROC OPTIONS (MAIN);
      DECLARE A CHAR(80);
      ON ENDFILE (INPUT) GOTO STOP;
LOOP: GET FILE (INPUT) EDIT (A) (A(80));
      PUT FILE (SYSPRINT) SKIP EDIT (A) (A);
      GOTO LOOP;
STOP: CLOSE FILE (INPUT);
      END PGM;
```

The program is stored in the cataloged library HIS.LOADTST, under the name PRNT. The program is then executed by:

```
// EXEC HIS1
//SYSIN DD *
:PRNT
/*
//INPUT DD *
    cards to be printed
/*
```

Programs that require an instruction -- If the program is going to accept the instruction passed by the supervisor, it must be written in either BAL or PL/I. If the program is in BAL, the instruction is retrieved by following the pointer in register 1 (see Figure 2-I-2). To retrieve the instruction in a PL/I program, a 100-character variable is coded on the PROC statement. Individual characters are then retrieved by overlay defining or by using SUBSTR. The preceding PL/I example program is to be modified such that the name of the input DD statement is to be specified on the instruction. When executing the modified program, the parameter DDNAME=ddname is specified on the command; the command decoder places the name in columns 24-31 of the instruction. A modified program listing is:

```
PGM:   PROC (PARM) OPTIONS (MAIN);
       DECLARE
          A CHAR(80),
          PARM CHAR(100),
          DDNAME CHAR(8) DEF PARM POS(24);
       ON ENDFILE (INPUT) GOTO STOP;
       OPEN FILE (INPUT) TITLE (DDNAME);
LOOP:  GET FILE (INPUT) EDIT (A) (A(80));
       PUT FILE (SYSPRINT) SKIP EDIT (A) (A);
       GOTO LOOP;
STOP:  CLOSE FILE (INPUT);
       END PGM;
```

The program may then be executed by:

```
// EXEC HIS1
//SYSIN DD *
:PRNT,DDNAME=OPERA
/*
//OPERA DD *
     cards for printing
/*
```

PL/I routines whose output is to be based on the HIS formatting options may utilize the printer subroutine. Assume that the above program is to be modified to use this subroutine. Assume also that the first card read is to be used as a heading printed on each page of the output. A blank line is to be placed between the heading and the first line of output on each page. A new listing that will accomplish this is:

```
PGM:   PROC (PARM) OPTIONS (MAIN);
       DECLARE
          PARM CHAR(100),
          DDNAME CHAR(8) DEF PARM POS(24),
          INSTR CHAR(80) EXTERNAL,
          #_HDGS PIC'Z' DEF INSTR POS(72),
          PRINTER CHAR(132) EXTERNAL,
          HEADING(9) CHAR(132) EXTERNAL,
          PRINTX ENTRY (PIC'Z'),
          A CHAR(80);
       ON ENDFILE (INPUT) GOTO STOP;
       OPEN FILE (INPUT) TITLE (DDNAME);
       CALL INIT (PARM);
       #_HDGS = 2;
       GET FILE (INPUT) EDIT (A) (A(80));
       HEADING(1) = A;
LOOP:  GET FILE (INPUT) EDIT (A) (A(80));
       PRINTER =A;
       CALL PRINTX (1);
       GOTO LOOP;
STOP:  CLOSE FILE (INPUT);
       CALL EXIT (PARM);
       END PGM;
```

The program may then use formatting options and be executed by:

```
// EXEC HIS1
//SYSIN DD *
:PRNT,DDNAME=OUCH,PAGE=NUMBER=1
/*
//OUCH DD *
  heading card
  cards for printing
/*
```

HIS.TABLES

HIS.TABLES is a cataloged library containing several tables used by HIS routines. These tables can be used to obtain county names from county numbers, and other similar functions. Each table is stored in 80-byte records, and may be updated with IBM utility program IEBUPDTE.

PGMTBL -- PGMTBL is a table of program names processed by the command decoder. Rather than specifying the actual load module name (1-4 characters) of a program, a program name (1-20 characters) may be assigned, and an entry placed in the program table. The decoder compares the program name specified on a command with the entries in the table, and, if a match is found, substitutes the load module name. If no match is found, the decoder uses the name specified on the command as the load module name (using the first four characters if the name is longer than 4 characters). Each record in the table specifies a program name in columns 1-20, and the load module name in columns 21-24. Columns 25-72 may contain any descriptive information desired; these are not processed. Columns 73-80 contain sequence numbers for updating the file. Programs may have alias names if desired; simply place additional records in the table, each specifying the same load module name.

CITYTBL -- CITYTBL provides the names of the incorporated cities of Montana, along with their population and county location. The table consists of a record for each city, located in the table by city number. Hence, the 35th record in the file corresponds to city number 35, Drummond. Each record contains the following information:

Columns 1-18:	City name, left justified.
Columns 19-36:	City name, centered.
Columns 37-54:	City name, left justified, with blanks replaced with hyphens.
Column 55:	Population group.
Column 56:	Blank.
Columns 57-58:	County number, alphabetical numbering.
Column 59:	Blank.
Columns 60-61:	County number, license numbering.
Columns 62-72:	Blank.
Columns 73-80:	Sequence number.

CNTYTBL -- CNTYTBL provides the name and financial district of each of the counties. The 56 records are placed in order by the alphabetical numbering system, placing Beaverhead as the first county. Each record contains:

Columns 1-15: County name, left justified.
Columns 16-30: County name, centered.
Columns 31-45: County name, right justified.
Columns 46-49: Blank.
Columns 50-51: Financial District (1-12).
Columns 52-72: Blank.
Columns 73-80: Sequence number.

PROJTBL -- PROJTBL is a table of the project types used in the Roadlog. The records are placed in order by classification number. Hence, class FU (classification number 5) is the 5th record. The records contain:

Columns 1-5: Project class, left justified.
Columns 6-10: Project class, centered.
Columns 11-15: Project class, right justified.
Columns 16-72: Blank.
Columns 73-80: Sequence number.

Project classes are often added to the system; hence, the table will contain an unknown number of records. Programs utilizing the table should be written to accept whatever number of classifications are available in the file. Because the records are stored by classification number, care must be taken when updating. New classifications should be placed after the last one in the file. When a classification is deleted, the record cannot be deleted from the file unless a dummy record (or a new classification) is stored in its place.

SURFTBL -- SURFTBL provides a method of converting from the 4-digit Roadlog surface type code to a 1-digit (1-8) conversion. Each surface type is coded on a separate record:

Columns 1-4: 4-digit surface type code.
Columns 5-8: 1-digit code, right justified.
Columns 9-10: Blank.
Columns 11-13: 3-character code indicating type group.
Columns 14-16: Blank.

Columns 17-26: 10-character heading.
Columns 27-36: 10-character heading.
Columns 37-72: Blank.
Columns 73-80: Sequence number.

SMSFTBL -- SMSFTBL provides a second table of surface types, used in producing state mileage tables. The records are of the same format as in SURFTBL, but provide a different breakdown of surface types and a different set of headings for printing reports. The Roadlog program SURF-TYPE reads this table instead of SURFTBL if REPORT=SMTABLES is specified in place of REPORT=ROADLOG.

SUFFTBL -- SUFFTBL contains a number of tables used in producing the Sufficiency-by-Sections report. The table is described in Chapters 1-V and 2-V.

CHAPTER 2-II
ROADLOG PROGRAMMER INFORMATION

Introduction

This chapter presents a description of the programs comprising the Roadlog subsystem of HIS (Highway Information System). It is designed for utilization with the publication Highway Information System Volume 1: User Information.

Roadlog File Description

Data Set Name	HIS.ROADLOG
Organization	Indexed Sequential
Logical Record Length	120
Physical Record Length	1560
Key Length	13
Volume Serial Number	231428

The internal format of a Roadlog record is shown in PL/I terminology in Figure 2-II-1. Most of the numeric fields are stored in packed decimal format to conserve storage and to improve efficiency.

The two location code fields are stored numerically rather than in the 4-byte character string coded on the Roadlog data cards. Table 2-II-I shows the relationship between these character strings and the stored values.

The project classification field and the surface type field have been added to the record to increase the efficiency of several summary and report programs. Tables 2-II-II and 2-II-III list the project classification and surface type codes.

Subroutines

Several of the Roadlog programs utilize the subroutines SRTYPR and RLGCON. The subroutines are stored, in object module format, in cataloged library HIS.OBJECT.

```
1   ROADLOG_RECORD,
    2  DELETE_CHARACTER                    CHAR(1),
    2  KEY,
       3  ROUTE_SYSTEM                     CHAR(1),
       3  ROUTE_NUMBER                     CHAR(3),
       3  REFERENCE_POST                   CHAR(3),
       3  DISTANCE                         CHAR(6),
    2  REMARK                              CHAR(2),
    2  SECTION_LENGTH                      DEC FIXED (5,3),
    2  ROUTE_LENGTH                        DEC FIXED (5,3),
    2  CONSTRUCTED_LENGTH                  DEC FIXED (5,3),
    2  UNIMPROVED_LENGTH                   DEC FIXED (5,3),
    2  WYE_LENGTH                          DEC FIXED (3,3),
    2  DESCRIPTION                         CHAR(35),
    2  PROJECT_NUMBER                      CHAR(11),
    2  DIVIDED_UNDIVIDED_CODE              CHAR(1),
    2  NUMBER_OF_LANES                     DEC FIXED (1,0),
    2  POPULATION_CODE                     DEC FIXED (1,0),
    2  CITY_NUMBER                         DEC FIXED (3,0),
    2  COUNTY_NUMBER                       DEC FIXED (2,0),
    2  YEAR_BUILT                          DEC FIXED (2,0),
    2  YEAR_IMPROVED                       DEC FIXED (2,0),
    2  FOREST_HIGHWAY_NUMBER               DEC FIXED (2,0),
    2  ADMINISTRATION_CODE                 DEC FIXED (2,0),
    2  LOCATION_CODES(2)                   DEC FIXED (2,0),
    2  PROJECT_CLASSIFICATION              DEC FIXED (2,0),
    2  SURFACE_WIDTH                       DEC FIXED (2,0),
    2  ROADWAY_WIDTH                       DEC FIXED (2,0),
    2  SURFACE_THICKNESS                   DEC FIXED (3,1),
    2  BASE_THICKNESS                      DEC FIXED (3,1),
    2  SURFACE_TYPE_CODE                   DEC FIXED (1,0),
    2  SURFACE_TYPE                        DEC FIXED (4,0),
    2  MAINTENANCE_SECTION                 DEC FIXED (4,0),
    2  DATE,
       3  MONTH                            DEC FIXED (2,0),
       3  DAY                              DEC FIXED (2,0),
       3  YEAR                             DEC FIXED (2,0),
    2  DUMMY                               CHAR(2);
```

Figure 2-II-1. Roadlog file structure.

TABLE 2-II-I

LOCATION CODES

Character	Numeric
blank	0
CITY	1
CNTY	2
NFOR	3
IRES	4
GAME	5
MRES	6
NMON	7
NPRK	8
SFOR	9
SPRK	10

TABLE 2-II-II

PROJECT CLASSIFICATIONS

1	F	18	ERS	35	CCC	52	WPMH
2	FI	19	CC	36	DARM	53	WPMS
3	FG	20	MC	37	ECHP	54	WPSO
4	FGI	21	NFD	38	EFAP	55	WPSS
5	FU	22	IN	39	EFHP	56	WPGH
6	S	23	USG	40	FAP	57	WPGM
7	SG	24	US	41	FAS	58	WPGS
8	U	25	I	42	FAGH	59	SC
9	UI	26	ING	43	FAGM	60	SAP
10	UG	27	DF	44	FAGS	61	WER
11	UGI	28	DFG	45	FLP	62	AE
12	FL	29	DS	46	NRFL	63	UR
13	FLI	30	DSG	47	NRH	64	UF
14	FLG	31	DU	48	NRM	65	WPH
15	FLGI	32	R-AD	49	NRS	66	A-AD
16	FHP	33	IG	50	NP		
17	ERF	34	ERFO	51	WHP		

TABLE 2-II-III

SURFACE TYPE CODES

Surface Type	Value
Primitive	1
Unimproved	2
Graded and Drained	3
Gravel	4
Bituminous Surface Treatment	5
Road Mix	6
Plant Mix	7
Portland Cement Concrete	8

SRTYPR --

Object Module Name	SRTYPR
Language	PL/I
Files	SYSPRINT -- IBM messages SURFTBL -- Surface Type Table
Entry Points	SRTYPRI SRTYPRA

SRTYPR is a subroutine which converts the 4-digit Roadlog surface type designation into a simpler 1-digit designation. The new designation is a number between 1 and 8 shown in Table 2-II-III. A value of zero is returned when an undefined Roadlog surface type is given. The subroutine utilizes a table defining the valid Roadlog surface type codes. The table is read via DD statement SURFTBL one time only, and stored in core for future use. Each record in the table defines one surface type, together with the new 1-digit designation to be returned. The table read will normally be member SURFTBL in cataloged library HIS.TABLES. To utilize SRTYPR, the calling program must first invoke the subroutine at entry point SRTYPRI. Here, SRTYPR reads the surface type table and stores the values in static array (binary format) SURTYP. This array allows for a maximum of 50 records in the table (variable #_SURF_TYPES notes the actual number of records read). After invoking SRTYPRI, the calling program may perform conversions by invoking SRTYPRA, passing a binary (length 15) surface type designation. The subroutine compares the value to the designations read from the table, substituting the new designation for the value passed. If no match is found, a value of zero is returned. Note that the new designation is returned in the same parameter in which the old designation is passed, destroying the passed value. An example of SRTYPR usage is:

```
PROG:  PROCEDURE OPTIONS (MAIN);
       DECLARE
          I BINARY FIXED (15),
          SURF_TYPE DEC FIXED (4,0);
       CALL SRTYPRI;
LOOP:  GET FILE (SURFTYP) EDIT (SURF_TYPE) (F(4,0));
       I = SURF_TYPE;
       CALL SRTYPRA (I);
       PUT FILE (SYSPRINT) SKIP EDIT (I) (A);
       IF I=0 THEN GOTO STOP;
       GOTO LOOP;
STOP:  END PROG;
```

Following is the SRTYPR program listing:

```
SRTYPRI:  PROCEDURE;

  1: SRTYPRI:  PROCEDURE;


  2: /*** SUBROUTINE TO RETURN SURFACE TYPE CLASS *****/


  3: /***** DECLARATION OF VARIABLES (ALPHABETICAL ORDER) *****/

  4: DECLARE
  5:    #_SURF_TYPES BIN FIXED STATIC,
  6:    INSTR CHAR(80) EXT,
  7:    REPORT CHAR(1) DEF INSTR POS(8),
  8:    1  STYP,
  9:       3 (S1,S2) PIC'ZZZZ',
 10:       3  DUMMY CHAR(72),
 11:    SURTYP(50,2) BIN FIXED STATIC,
 12:    TABLE FILE INT RECORD;


 13: /***** PROGRAM INITIALIZATION SECTION *****/

 14: INIT:
 15:    IF REPORT¬='Q'
 16:       THEN OPEN FILE (TABLE) INPUT RECORD TITLE ('SURFTBL');
 17:       ELSE OPEN FILE (TABLE) INPUT RECORD TITLE ('SMSFTBL');
 18:    ON ENDFILE (TABLE) GOTO CONTINUE;
 19:    #_SURF_TYPES = 0;
 20:    SURTYP = 0;
 21: READ_TEMP:
 22:    READ FILE (TABLE) INTO (STYP);
 23:    #_SURF_TYPES = #_SURF_TYPES + 1;
 24:    IF #_SURF_TYPES>50 THEN GOTO CONTINUE;
 25:    SURTYP(#_SURF_TYPES,1) = STYP.S1;
 26:    SURTYP(#_SURF_TYPES,2) = STYP.S2;
 27:    GOTO READ_TEMP;
 28: CONTINUE:
 29:    CLOSE FILE (TABLE);
 30:    RETURN;


 31: /***** ENTRY POINT FOR DETERMINING ROADLOG SURFACE TYPE CATEGORY ****


 32: SRTYPRA:
 33:    ENTRY (J1);
 34:    DO J2=1 TO #_SURF_TYPES;
 35:       IF J1=SURTYP(J2,1) THEN GO TO SATISFIED_SURFACE_TYPE;
 36:       END;
 37:    J1 = 0;   /** NO MATCH **/
 38:    RETURN;
 39: SATISFIED_SURFACE_TYPE:
 40:    J1 = SURTYP(J2,2);
 41:    IF J1=9 THEN J1 = 7;
 42:    RETURN;

 43: END SRTYPRI;
```

RLGCON --

Object Module Name	RLGCON
Language	PL/I
Subroutines	SRTYPR
Files	SYSPRINT -- IBM messages PROJTBL -- Project Class Table SURFTBL -- Surface Type Table
Entry Points	RLGCON COV1 COV2

RLGCON converts Roadlog data cards into the Roadlog file format, and vice versa. When conversions from the Roadlog 2-card sequence to the Roadlog file format must be performed, the subroutine must first be invoked at initialization entry point RLGCON. At this entry point, the project class table (DD name PROJTBL) is read, and SRTYPR is invoked at its initialization entry point, SRTYPRI. The subroutine is then invoked at entry COV1 as many times as required, passing two 80-character data cards, and a 120-character variable for the returned Roadlog record. An example of such usage is:

```
PGM:   PROCEDURE OPTIONS (MAIN);
       DECLARE
          (CARD1,CARD2) CHAR(80),
          RECORD CHAR (120);
       CALL RLGCON;
LOOP:  GET FILE (SYSIN) EDIT (CARD1,CARD2) (A(80));
       CALL COV1 (CARD1,CARD2,RECORD);
       WRITE FILE (OUTPUT) FROM (RECORD);
       GOTO LOOP;
       END PGM;
```

When converting from Roadlog record format to card format, it is not necessary to first read the project and surface type tables. RLGCON is simply invoked at entry point COV2, passing two 80-character variables for the returned cards, and a 120-character Roadlog record (e.g., CALL COV2 (CARD1,CARD2,RECORD); in the above program).

The RLGCON program listing follows:

RLGCON: PROCEDURE;

```
 1:  RLGCON: PROCEDURE;
 2:          DECLARE
 3:           1 CD1 BASED(PTR_CD1),
 4:             2 DUM1 CHAR(1),
 5:             2 KEY CHAR(13),
 6:             2 DESCR CHAR(35),
 7:             2 MAINT_SECTN PIC'ZZZZ',
 8:             2 PROJ_# CHAR(11),
 9:             2 REMARK CHAR(2),
10:             2 (CNTY_#,FORHWY_#,ADMIN) PIC'ZZ',
11:             2 LOCN(2) CHAR(4),
12:           1 CD2 BASED(PTR_CD2),
13:             2 DUM1 CHAR(1),
14:             2 KEY CHAR(13),
15:             2 #_LANES PIC'Z',
16:             2 DIVIDED CHAR(1),
17:             2 POPULATION PIC'Z',
18:             2 CITY_# PIC'ZZZ',
19:             2 YR_BLT PIC'ZZ',
20:             2 SURF_TYPE PIC'ZZZZ',
21:             2 SURF_THICK PIC'ZVZ',
22:             2 BASE_THICK PIC'ZZVZ',
23:             2 (SURF_WIDTH,ROAD_WIDTH) PIC'ZZ',
24:             2 (ROUTE,CNST,UNIMP) PIC'ZZVZZZ',
25:             2 WYE PIC'VZZZ',
26:             2 SECTN PIC'ZZVZZZ',
27:             2 YR_IMP PIC'ZZ',
28:             2 DATE,
29:               3 (MONTH,DAY,YEAR) PIC'ZZ',
30:           1 RLG BASED(PTR_NEW),
31:             2 DUMMY1 CHAR(1),
32:             2 KEY CHAR(13),
33:             2 REMARK CHAR(2),
34:             2 (SECTN,ROUTE,CONST,UNIMP)DEC FIXED(5,3),
35:             2 WYE DEC FIXED(3,3),
36:            2 DESCR CHAR(35),
37:             2 PROJ_# CHAR(11),
38:             2 DIVIDED CHAR(1),
39:             2 (#_LANES,POPULATION)DEC FIXED(1,0),
40:             2 (CITY_#,CNTY_#,YR_BLT,YR_IMP,FORHWY_#,ADMIN,
41:               LOCATION(2),PROJ_CLASS,SURF_WIDTH,ROAD_WIDTH)
42:               DEC FIXED(3,0),
43:             2 (SURF_THICK,BASE_THICK)DEC FIXED(3,1),
44:             2 SURF_TYPE_CODE DEC FIXED(1,0),
45:             2 (SURF_TYPE,MAINT_SEC)DEC FIXED(5,0),
46:             2 DATE,
47:               3 (MONTH,DAY,YEAR)DEC FIXED(3,0),
48:           PROJTBL FILE RECORD,
49:           PROJ_TYPE(75) CHAR(4) STATIC,
50:           #_PROJ BIN FIXED,
51:           PRJ CHAR(4) BASED(PTR),
52:           (CARD1,CARD2) CHAR(80),
53:           RECORD CHAR(120),
54:           D3 DEC FIXED(3,0),
55:           (A1,D4) CHAR(4);
56:  /**INITIALIZATION**/
57:          OPEN FILE (PROJTBL);
58:          ON ENDFILE (PROJTBL) GOTO END_PROJ;
```

```
RLGCON: PROCEDURE;

 59:          DO #_PROJ = 1 TO 75;
 60:             READ FILE (PROJTBL) SET(PTR);
 61:             PROJ_TYPE (#_PROJ)= PRJ;
 62:          END;
 63:          END_PROJ:
 64:             #_PROJ=#_PROJ-1;
 65:             CLOSE FILE(PROJTBL);
 66:          CALL SRTYPRI;
 67:          RETURN;
 68:     COV1: ENTRY(CARD1,CARD2,RECORD);
 69:          PTR_CD1=ADDR(CARD1);
 70:          PTR_CD2=ADDR(CARD2);
 71:          PTR_NEW=ADDR(RECORD);
 72:          RECORD='     ';
 73:          RLG=CD1,BY NAME;
 74:          RLG=CD2,BY NAME;
 75:          DO I= 1 TO 2;
 76:          D4=CD1.LOCN(I);
 77:          IF D4=' ' THEN D3=0;
 78:          ELSE IF D4='CITY' THEN D3=1;
 79:          ELSE IF D4='CNTY' THEN D3=2;
 80:          ELSE IF D4='NFOR' THEN D3=3;
 81:          ELSE IF D4='IRES' THEN D3=4;
 82:          ELSF IF D4='GAME' THEN D3=5;
 83:          ELSE IF D4='MRES' THEN D3=6;
 84:          ELSE IF D4='NMON' THEN D3=7;
 85:          ELSE IF D4='NPRK' THEN D3=8;
 86:          ELSE IF D4='SFOR' THEN D3=9;
 87:          ELSE IF D4='SPRK' THEN D3=10;
 88:          ELSE D3=11;
 89:          RLG.LOCATION(I)=D3;
 90:          END;
 91:          I=RLG.SURF_TYPE;
 92:          CALL SRTYPRA(I);
 93:          RLG.SURF_TYPE_CODE=I;
 94:          A1=RLG.PROJ_#;
 95:          IF SUBSTR(A1,2,1)=' ' THEN SUBSTR(A1,3)=' ';
 96:          IF SUBSTR(A1,3,1)=' ' THEN SUBSTR(A1,4)=' ';
 97:          DO I=1 TO #_PROJ;
 98:             IF A1=PROJ_TYPE(I) THEN GOTO VAR3;
 99:          END;
100:          RLG.PROJ_CLASS=0;
101:          GO TO VAR4;
102:     VAR3: RLG.PROJ_CLASS=I;
103:     VAR4: RETURN;
104:     COV2: ENTRY(CARD1,CARD2,RECORD);
105:          PTR_CD1=ADDR(CARD1);
106:          PTR_CD2=ADDR(CARD2);
107:         PTR_NEW=ADDR(RECORD);
108:         CARD1=' ';
109:         CARD2=' ';
110:          CD2=RLG,BY NAME;
111:          CD2.DUM1='2';
112:          CD1=RLG, BY NAME;
113:          CD1.DUM1='1';
114:          DO I=1 TO 2;
115:          D3=RLG.LOCATION(I);
116:          IF D3=0 THEN D4=' ';
```

```
RLGCON: PROCEDURE;

117:            ELSE IF D3=1 THEN D4='CITY';
118:            ELSE IF D3=2 THEN D4='CNTY';
119:            ELSE IF D3=3 THEN D4='NFOR';
120:            ELSE IF D3=4 THEN D4='IRES';
121:            ELSE IF D3=5 THEN D4='GAME';
122:            ELSE IF D3=6 THEN D4='MRES';
123:            ELSE IF D3=7 THEN D4='NMON';
124:            ELSE IF D3=8 THEN D4='NPRK';
125:            ELSE IF D3=9 THEN D4='SFOR';
126:            ELSE IF D3=10 THEN D4='SPRK';
127:            ELSE D4='ERR ';
128:            CD1.LOCN(I)=D4;
129:            END;
130:            RETURN;
131:            END RLGCON;
```

Program Descriptions

Each program in the Roadlog subsystem is stored in load module format in cataloged library HIS.LOADLIB, from which it is retrieved for execution by the HIS supervisor when requested. The member name for each program is given with the program description.

This section of the manual presents a write-up on each program in the Roadlog subsystem. An attempt has been made in the source listing itself to document the programs with appropriate variable names and comments.

DUMP --

Member Name	DMR	
Language	PL/I	
Subroutines	PRINTX1	
	RLGCON	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- "dump" listing
	ROADLOG	-- Roadlog file
Instruction	1 - 3	"DMR"
	40 - 52	Beginning key
	56 - 68	Ending key

DUMP provides an unformatted listing of data in the Roadlog file, between user-specified (by means of the DATA parameter) records in the file. Subroutine RLGCON is used to convert the Roadlog records into character format for printing. Subroutine PRINTX1 is used for printer output, allowing the use of HIS formatting options in printing. RLGCON is never invoked at its initialization entry point, as the project and surface type tables are not required when converting Roadlog records into data card format. Hence, the subroutine SRTYPR (used only by RLGCON entry points RLGCON and COV1) does not need to be linked into the load module. When linking without this subroutine, the LET option must be specified to the link editor.

The DMR program listing follows:

/* ROADLOG FILE DUMP */

```
 1: /* ROADLOG FILE DUMP */
 2: DUMP: PROC(PARM) OPTIONS(MAIN);
 3: /* INSTRUCTION AND PRINT ROUTINE */
 4: DCL
 5:   PARM CHAR(100),
 6:   INSTR CHAR(80) EXT,
 7:   STARTKEY CHAR(13) DEF INSTR POS(40),
 8:   ENDKEY CHAR(13) DEF INSTR POS(56),
 9:   #_HDGS PIC'Z' DEF INSTR POS(72),
10:   (PRINTER,HEADING(9)) CHAR(132) EXT,
11:   PRINTX ENTRY (PIC'Z');
12: /* DATA INPUT */
13: DCL
14:   RECORD CHAR(120) BASED(PTR_RLG),
15:   (CARD1,CARD2) CHAR(80),
16:   ROADLOG FILE RECORD KEYED INPUT SEQL ENV(INDEXED);
17: /*****   INITIALIZATION   *****/
18:   CALL INIT(PARM);
19:   #_HDGS=2;
20:   HEADING(1)='         ROADLOG DUMP';
21:   OPEN FILE(ROADLOG);
22:   ON ENDFILE(ROADLOG) BEGIN;
23:     PRINTER='   END OF FILE.';
24:     CALL PRINTX(3);
25:     GOTO DONE;
26:     END;
27:   READ FILE(ROADLOG) SET(PTR_RLG) KEY(STARTKEY);
28: /*****   EXECUTION LOOP   *****/
29: LOOP:
30:    CALL COV2 (CARD1,CARD2,RECORD);
31:   PRINTER=SUBSTR(CARD1,2)||SUBSTR(CARD2,15);
32:   CALL PRINTX(1);
33:   READ FILE (ROADLOG) SET(PTR_RLG);
34:   IF SUBSTR(RECORD,2,13)<=ENDKEY THEN GOTO LOOP;
35: DONE:
36:   CLOSE FILE(ROADLOG);
37:   CALL EXIT(PARM);
38: END DUMP;
```

LIST --

Member Name	PFR	
Language	PL/I	
Subroutines	PRINTX1	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- Roadlog listing
	ROADLOG	-- Roadlog file
Instruction	1 - 3	"PFR"
	40 - 52	Beginning key
	56 - 68	Ending key

LIST provides a listing of Roadlog data, in a more easily read formatted version than that provided by DUMP. Data conversions are performed within LIST, rather than by RLGCON. Not all of the Roadlog data items can be shown in the listing. Those to be shown are declared in the output structure, FRLG. The items to be printed are automatically converted and moved into this structure when FRLG is set equal to the Roadlog structure RLG by name. The section length is also accumulated (the accumulated total set to zero at the beginning of each route processed), and the accumulated value printed. In order to accumulate the section length, coincident sections must be scanned. Each time a "CO" record is found, the beginning and ending key of the coincident section is formed from the description, and the records retrieved. Although the records are not printed, the section length on each record in the coincident section is added to the total.

The PFR program listing follows:

```
/* :LIST,FILE=ROADLOG,DATA=XXXXX */

  1: /* :LIST,FILE=ROADLOG,DATA=XXXXX */

  2: LIST:  PROCEDURE (PARM) OPTIONS (MAIN);


  3: /***** DECLARATION OF VARIABLES (ALPHABETICAL ORDER) *****/

  4: DECLARE
  5:    #_HDGS PIC'Z' DEF INSTR POS(72),
  6:    ACCUM_MILEAGE PIC'ZZZVZZZ',
  7:    CODE(0:10) CHAR(4) STATIC INIT
  8:       ('    ','CITY','CNTY','NFOR','IRES','GAME',
  9:        'MRES','NMON','NPRK','SFOR','SPRK'),
 10:    COIN_IND CHAR(2),
 11:    DRLG CHAR(120) STATIC,
 12:    DFRLG CHAR(132) DEF FRLG,
 13:    ENDKEY CHAR(13) DEF INSTR POS(56),
 14:    F(0:9) PIC'Z' STATIC INIT (0,1,2,3,4,5,6,7,8,9),
 15:    1  FRLG STATIC,
 16:       3  RT_# CHAR(5),
 17:       3  MILEPOST CHAR(10),
 18:       3  DESCR CHAR(36),
 19:       3  PROJ_CLASS CHAR(5),
 20:       3  YR_BLT PIC'ZZB',
 21:       3  SURTYP CHAR(4),
 22:       3  SURF_WIDTH PIC'ZZ',
 23:       3  RDWAY_WIDTH PIC'ZZZ',
 24:       3  LANES PIC'ZZB',
 25:       3  LOCN1 CHAR(5),
 26:       3  LOCN2 CHAR(5),
 27:       3  ROUTE PIC'ZZV.ZZZ',
 28:       3  SECTN PIC'ZZZV.ZZZ',
 29:       3  CONST PIC'ZZZV.ZZZ',
 30:       3  UNIMP PIC'ZZZV.ZZZ',
 31:       3  WYE PIC'ZV.ZZZ',
 32:       3  ACCUM_MLGE PIC'ZZZZV.ZZZB',
 33:       3  REMARK CHAR(3),
 34:       3  COUNTY_# PIC'ZZ',
 35:       3  CITY_# PIC'ZZZZB',
 36:    HEADING(9) CHAR(132) EXT,
 37:    INSTR CHAR(80) EXT,
 38:   (KEY1,KEY2) CHAR(13),
 39:    PARM CHAR(100),
 40:    PRINTER CHAR(132) EXT,
 41:    1 RLG BASED(PTR_DRLG),
 42:       3  DUMMY1 CHAR(1),
 43:       3  RT_# CHAR(4),
 44:       3 MILEPOST CHAR(9),
 45:       3 REMARK CHAR(2),
 46:       3 (SECTN,ROUTE,CONST,UNIMP) DEC FIXED(5,3),
 47:       3 WYE DEC FIXED(3,3),
 48:       3 DESCR CHAR(35),
 49:       3 PROJ_CLASS CHAR(11),
 50:       3 DIVIDED CHAR(1),
 51:       3 (LANES,POPULATION_CODE) DEC FIXED(1,0),
 52:       3 (CITY_#,COUNTY_#,YR_BLT,YR,FORHWY_#,ADMIN_CODE,
 53:          LOC1,LOC2,PROJ_CL,SURF_WIDTH,RDWAY_WIDTH)
 54:          DEC FIXED(3,0),
```

```
/* :LIST,FILE=ROADLOG,DATA=XXXXX */

 55:       3 (SURF_THICKNESS,BASE_THICKNESS) DEC FIXED(3,1),
 56:       3 SURF_TYPE_CODE DEC FIXED(1,0),
 57:       3 (SURF_TYPE,MAINT_SEC) DEC FIXED(5,0),
 58:       3 DATE,
 59:          5 (MONTH,DAY,YEAR)  DEC FIXED(3,0),
 60:    RLGKEY CHAR(13) DEF DRLG POS(2),
 61:    ROADLOG FILE RECORD KEYED ENV(INDEXED),
 62:    SAVE_KEY CHAR(13),
 63:    SAVE_RT_# CHAR(4),
 64:    STARTKEY CHAR(13) DEF INSTR POS(40),
 65:    TYPE(0:8) CHAR(3) STATIC INIT
 66:       ('   ','PRM','BLD','GRD','GRV','BST','RMS','PMS','PCC');


 67: /***** PROGRAM INITIALIZATION *****/

 68:    CALL INIT (PARM);

 69:    /*** SET UP PAGE HEADINGS ***/
 70:    #_HDGS = 3;
 71:    HEADING(1) = '         MILE                                          '
 72:       || '    YR SUR WIDTH                ' ||
 73:       '************ LENGTH ************  ACCUM';
 74:    HEADING(2) = 'RT_#    POST    ******* SECTION DESCRIPTION ********'
 75:       || 'PROJ BT TYP SF RD L LOCATIONS ' ||
 76:       ' ROUTE  SECTN  CONST  UNIMP  WYE MILEAGE RE CY CITY';

 77:    /*** INITIALIZF ROADLOG FILE ***/
 78:    OPEN FILE (ROADLOG) INPUT;
 79:    ON ENDFILE (ROADLOG) GOTO RETURN;
 80:    READ FILE (ROADLOG) INTO (DRLG) KEY (STARTKEY);
 81:    PTR_DRLG=ADDR(DRLG);
 82:    SAVE_RT_# = ' ';


 83: /***** MAIN CONTROL SECTION *****/

 84: LOOP:
 85:    IF RLG.RT_#¬=SAVE_RT_# THEN DO;
 86:       ACCUM_MILEAGE = 0;
 87:       FRLG.ACCUM_MLGE = 0;
 88:       J = 3;
 89:       SAVE_RT_# = RLG.RT_#;
 90:       END;
 91:     FRLG=RLG, BY NAME;
 92:    FRLG.LOCN1 = CODE(RLG.LOC1);
 93:    FRLG.LOCN2 = CODE(RLG.LOC2);
 94:    DO J1=3 TO 4;
 95:       IF SUBSTR(FRLG.PROJ_CLASS,J1-1,1)=' ' THEN
 96:          SUBSTR(FRLG.PROJ_CLASS,J1,1) = ' ';
 97:       END;
 98:    FRLG.SURTYP = TYPE(RLG.SURF_TYPE_CODE);
 99:    ACCUM_MILEAGE = ACCUM_MILEAGE + RLG.SECTN;
100:    IF RLG.REMARK=' ' | RLG.REMARK='SP' | RLG.REMARK='LP' |
101:       RLG.REMARK='NE' | RLG.REMARK='OS' THEN
102:       FRLG.ACCUM_MLGE = ACCUM_MILEAGE;
103:    ELSE IF RLG.REMARK='CO' | RLG.REMARK='IL'
104:       THEN CALL CO_PROCESSOR;
```

```
/* :LIST,FILE=ROADLOG,DATA=XXXXX */

105:     PRINTER = DFRLG;
106:     CALL PRINTX (F(J));
107:     J = 1;
108:     READ FILE (ROADLOG) INTO (DRLG);
109:     IF RLGKEY<=ENDKEY THEN GOTO LOOP;


110: RETURN:
111:     CLOSE FILE (ROADLOG);
112:     CALL EXIT (PARM);
113:     RETURN;


114: /***** SUBROUTINE TO PROCESS COINCIDENT SECTIONS *****/

115: CO_PROCESSOR:  PROCEDURE;
116:     COIN_IND = RLG.REMARK;
117:     KEY1 = SUBSTR(RLG.DESCR,6,4) || SUBSTR(RLG.DESCR,16,9);
118:     KEY2 = SUBSTR(RLG.DESCR,6,4) || SUBSTR(RLG.DESCR,26,9);
119:     SAVE_KEY = RLGKEY;
120:     ON KEY (ROADLOG) BEGIN;
121:        CALL ASTER;
122:        PRINTER = '*** NO RECORD FOR KEY ' || KEY1;
123:        CALL PRINTX (F(2));
124:        CALL ASTER;
125:        GOTO BACK;
126:        END;
127:     READ FILE (ROADLOG) INTO (DRLG) KEY (KEY1);
128:     KEY1 = KEY2;
129:     KEY2 = RLGKEY;
130:     READ FILE (ROADLOG) INTO (DRLG) KEY (KEY1);
131:     IF COIN_IND='IL' THEN GOTO BACK;
132:     KEY1 = KEY2;
133:     KEY2 = RLGKEY;
134:     READ FILE (ROADLOG) INTO (DRLG) KEY (KEY1);
135:     DO WHILE (RLGKEY<KEY2);
136:        ACCUM_MILEAGE = ACCUM_MILEAGE + RLG.SECTN;
137:        READ FILE (ROADLOG) INTO (DRLG);
138:        END;
139:     FRLG.ACCUM_MLGE = ACCUM_MILEAGE;

140: BACK:
141:     READ FILE (ROADLOG) INTO (DRLG) KEY (SAVE_KEY);
142:     END CO_PROCESSOR;

143: END LIST;
```

LIST-ILOOPS --

Member Name	LIR	
Language	PL/I	
Subroutines	PRINTX1	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- LIST-ILOOPS output
	ROADLOG	-- Roadlog file
Instruction	1 - 3	"LIR"

LIST-ILOOPS prints a listing of all of the Interstate loops in the Roadlog file. An Interstate loop is a stretch of Interstate highway parallel to a Primary route. A record containing remark code "IL" appears, in the Primary route, defining the beginning and ending of a loop. The program scans the entire Primary route system for "IL" records, forms the beginning and ending keys from the description, and lists the records in the loop. Mileage is accumulated for each loop, and the mileage printed after the loop listing. After the entire listing, the total Interstate loop mileage is printed. The purpose of LIST-ILOOPS is to aid in the production of the Interstate and Primary summary by location in the Federal Aid Roadlog report. In this summary, the status of the 7% system is reported. The 7% system allows for subtraction of Interstate loops which are located outside Federal reservations and are not urban (urban loops are deducted when urban extensions are taken out). Hence, the program shows the mileage in three categories: that which is urban, that which is non-urban outside Federal reservations, and all other. Each record is flagged with one of the codes "URB," "OUT," and blank to indicate the mileage type of that record.

The LIR program listing follows:

```
/* :LIST_ILOOPS,FILE=ROADLOG */

  1: /* :LIST_ILOOPS,FILE=ROADLOG */
  2: ILOOP: PROC(PARM) OPTIONS(MAIN);
  3: /* INSTRUCTIONS */
  4: DCL
  5:  INSTR CHAR(80) EXT,
  6:  #_HOGS PIC'Z' DEF INSTR POS(72);
  7: /* PRINT SUBROUTINE */
  8: DCL
  9:  PARM CHAR(100),
 10:  HEADING(9) CHAR(132)EXT,
 11:  PRINTER CHAR(132)EXT,
 12:  PRINTX ENTRY(PIC'Z'),
 13:  PRINTXA ENTRY(PIC'Z',PIC'ZZ');
 14: /* ROADLOG FILE */
 15: DCL
 16:  1 RLG BASED(PTR_RLG),
 17:    3 DUM1 CHAR(2),
 18:    3 (RT_#,MILEPOST) PIC'ZZZ',
 19:    3 OFFSET CHAR(6),
 20:    3 REMARK CHAR(2),
 21:    3 (SECTN,ROUTE) DEC FIXED(5,3),
 22:    3 DUM2 CHAR(8),
 23:    3 DESCR CHAR(35),
 24:    3 PROJ CHAR(11),
 25:    3 DUM3 CHAR(2),
 26:    3 POP DEC FIXED(1,0),
 27:    3 (CITY_#,COUNTY_#,YR_BLT,YR_IMP,DUM6,DUMM7,LOC1,LOC2,
 28:      DUM7,S_WD,R_WD) DEC FIXED(3,0),
 29:    3 DUM8 CHAR(5),
 30:    3 (SURF_TYPE,DUM9) DEC FIXED(5,0),
 31:    3 DATE,
 32:     5 (MONTH,DAY,YEAR) DEC FIXED(3,0),
 33:    3 DUMMY2 CHAR(2),
 34:  1 RLGA BASED(PTR_RLG),
 35:    3 DUM1 CHAR(1),
 36:    3 KEY CHAR(13),
 37:    3 DUM2 CHAR(22),
 38:    3 RT_# CHAR(3),
 39:    3 DUM3 CHAR(6),
 40:    3 MP1 CHAR(9),
 41:    3 DUM4 CHAR(1),
 42:    3 MP2 CHAR(9),
 43:  ROADLOG FILE RECORD KEYED ENV(INDEXED);
 44: /* OUTPUT STRUCTURE */
 45: DCL
 46:  OUT CHAR(132) STATIC INIT(' '),
 47:   1 O DEF OUT POS(8),
 48:    3 MILEPOST PIC'ZZZ',
 49:    3 OFFSET CHAR(8),
 50:    3 DESCR CHAR(37),
 51:    3 PROJ CHAR(13),
 52:    3 REMARK CHAR(4),
 53:    3 COUNTY_# PIC'ZZBB',
 54:     3 (LOCN1,LOCN2) CHAR(5),
 55:     3 POP PIC'ZZ',
 56:    3 CITY_# PIC'ZZZBB',
 57:    3 YR_BLT PIC'ZZB',
 58:    3 YR_IMP PIC'ZZBB',
```

```
/* :LIST_ILOOPS,FILE=ROADLOG */

  59:     3 SURF_TYPE PIC'ZZZZBB',
  60:     3 S_WD PIC'ZZB',
  61:     3 R_WD PIC'ZZBB',
  62:     3 ROUTE PIC'ZZV.ZZZBB',
  63:     3 SECTN PIC'ZZV.ZZZBB',
  64:     3 CODE CHAR(3);
  65: /* OTHER VARIABLES */
  66: DCL
  67:  (KEY1,KEY2) CHAR(13),
  68:  SAVEKEY CHAR(13),
  69:  CNTR PIC'ZZ9' STATIC INIT(0),
  70:  (MP1,MP2) CHAR(9),
  71:  X PIC'Z',
  72:  MILES(2,3) DEC FIXED (7,3) STATIC,
  73:  CODE(3) CHAR(3) STATIC INIT('OUT','URB',' ');
  74: /****INITIALIZATION ***/
  75:  CALL INIT (PARM);
  76: /* SET UP HEADINGS */
  77:  #_HDGS=5;
  78:  SUBSTR(HEADING(1),72)='ROADLOG INTERSTATE LOOPS';
  79:  HEADING(3)='           MILE                                          '
  80:   || '   PROJECT          CY              CITY  YEARS  SURF  WIDTH  '
  81:    || '*** LENGTH ***';
  82:  HEADING(4)='           POST         SECTION DESCRIPTION              '
  83:   || '   NUMBER      RE  NO   LOCATIONS NMBR  BT IM  TYPE  SF RD  '
  84:    || 'ROUTE  SECTION';
  85:  /* INIT FILE */
  86:  OPEN FILE(ROADLOG) INPUT SEQL;
  87:  READ FILE(ROADLOG) SET(PTR_RLG) KEY('P001');
  88:  ON KEY (ROADLOG) BEGIN;
  89:    PRINTER='*** NO ROADLOG RECORD FOR KEY ' || KEY1;
  90:    CALL PRINTX(3);
  91:    GOTO RESTART;
  92:    END;
  93:  MILES = 0;
  94: /***** MAIN EXECUTION LOOP ****/
  95:  LOOP:
  96:   IF RLG.REMARK¬='IL' THEN GOTO READ_RECORD;
  97:   CNTR=CNTR+1;
  98:   PRINTER='  INTERSTATE LOOP  #' || CNTR;
  99:   CALL PRINTXA(5,12);
 100:   PRINTER='     PRIMARY ROUTE '|| RLG.RT_#;
 101:   CALL PRINTX(2);
 102:   MP1=RLGA.MP1;
 103:   MP2=RLGA.MP2;
 104:   PRINTER='     LOOP ON INTERSTATE ROUTE ' || RLGA.RT_# ||
 105:     ' FROM MILEPOST '||MP1|| ' TO '|| MP2;
 106:   CALL PRINTX(1);
 107:  X=2;
 108:  MILES(1,*)=0;
 109:  KEY1=SUBSTR(RLG.DESCR,6,4)||SUBSTR(RLG.DESCR,16,9);
 110:  KEY2=SUBSTR(RLG.DESCR,6,4)||SUBSTR(RLG.DESCR,26,9);
 111:  SAVEKEY=RLGA.KEY;
 112:  READ FILE(ROADLOG) SET(PTR_RLG) KEY(KEY1);
 113:  DO WHILE(RLGA.KEY<KEY2);
 114:       IF RLG.POP>=4 THEN I = 2;
 115:       ELSE IF RLG.LOC2¬=0 & RLG.LOC2¬=9 |
 116:          RLG.LOC1¬=1 & RLG.LOC1¬=2 & RLG.LOC1¬=9 THEN I = 3;
```

```
/* :LIST_ILOOPS,FILE=ROADLOG */

117:        ELSE I = 1;
118:    MILES(1,I)=MILES(1,I) + RLG.SECTN;
119:    OUT = ' ';
120:    O=RLG,BY NAME;
121:    IF RLG.REMARK=' ' | RLG.REMARK='SP' | RLG.REMARK='LP' |
122:        RLG.REMARK='NE' | RLG.REMARK='OS' THEN O.CODE = CODE(I);
123:    PRINTER=OUT;
124:    CALL PRINTX(X);
125:    X=1;
126:    READ FILE(ROADLOG) SET(PTR_RLG);
127:    END;
128:  RESTART: READ FILE(ROADLOG) SET(PTR_RLG) KEY(SAVEKEY);
129:     MILES(2,*) = MILES(2,*) + MILES(1,*);
130:    PRINTER='              *** MILEAGES FOR LOOP ***';
131:    CALL PRINTXA(3,10);
132:    PRINTER='      OUTSIDE FEDERAL RESERV.  ' || MILES(1,1);
133:     CALL PRINTX(2);
134:     PRINTER='                              URBAN  ' || MILES(1,2);
135:    CALL PRINTX(1);
136:    PRINTER='                         ALL OTHER  ' || MILES(1,3);
137:    CALL PRINTX(1);
138:    MILES(1,3)=MILES(1,1)+MILES(1,2)+MILES(1,3);
139:    PRINTER='                               TOTAL  ' || MILES(1,3);
140:    CALL PRINTX(2);
141: READ_RECORD:
142:    READ FILE(ROADLOG) SET (PTR_RLG);
143:    IF RLGA.KEY<'S' THEN GOTO LOOP;
144:    PRINTER='      TOTAL INTERSTATE LOOP MILEAGES';
145:     #_HDGS = 2;
146:     CALL PRINTXA (6,13);
147:    PRINTER='    OUTSIDE FEDERAL RESERV.   ' || MILES(2,1);
148:    CALL PRINTX(2);
149:    PRINTER='                              URBAN   ' || MILES(2,2);
150:    CALL PRINTX(1);
151:    PRINTER='                         ALL OTHER   ' || MILES(2,3);
152:    CALL PRINTX(1);
153:    MILES(2,3)=MILES(2,1)+MILES(2,2)+MILES(2,3);
154:    PRINTER='                               TOTAL  ' || MILES(2,3);
155:    CALL PRINTX(2);
156:    CLOSE FILE(ROADLOG);
157:    CALL EXIT (PARM);
158: END ILOOP;
```

UPDATE -- UPDATE is comprised of four separate programs, one for each of the functions DELETE, INSERT, REWRITE, and NEW-KEY. The names of the routines are "PDR" followed by the first letter of the function (e.g., "PDRD" for FUNCTION=DELETE).

FUNCTION=DELETE:

Member Name	PDRD	
Language	PL/I	
Subroutines	PRINTX1	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- UPDATE messages
	ROADLOG	-- Roadlog file
	any name	-- Roadlog data cards
Instruction	1 - 4	"PDRD"
	24 - 31	Name of input DD statement

The user codes, by means of the DDNAME parameter on the UPDATE command, the name of the DD statement he will supply with deletion data. Each card read via this DD statement contains, in columns 1-13, the key of a record to be deleted. The program operates with a direct update Roadlog file; each time a card is read, a PL/I DELETE statement specifying the key on the card is executed. If the key condition is raised (the specified record did not exist in the file), an error message is printed. Each data card is printed as it is read.

The PDRD program listing follows:

```
/* :UPDATE,FILE=ROADLOG,FUNCTION=DELETE,DDNAME=XXXXX */

 1: /* :UPDATE,FILE=ROADLOG,FUNCTION=DELETE,DDNAME=XXXXX */

 2: DELETE:  PROCEDURE (PARM) OPTIONS (MAIN);

 3: /* INSTRUCTION */
 4: DECLARE
 5:    INSTR CHAR(80) EXT,
 6:    #_HDGS PIC'Z' DEF INSTR POS(72),
 7:    DDNAME CHAR(8) DEF INSTR POS(24);

 8: /* PRINT ROUTINE */
 9: DECLARE
10:    PARM CHAR(100),
11:   (HEADING(9),PRINTER) CHAR(132) EXT,
12:    PRINTX ENTPY (PIC'Z');

13: /* PERMANENT FILE */
14: DECLARE
15:    PERMDD CHAR(8) STATIC INIT ('ROADLOG'),
16:    PERM FILE RECORD KEYED ENV (INDEXED);
17: DECLARE
18:    DATA FILE RECORD,
19:    KEY CHAR(13) BASED (PTR_DATA);


20: /***** INITIALIZATION *****/
21:    CALL INIT (PARM);

22:    /* SET UP HEADINGS */
23:    #_HDGS = 2;
24:    HEADING(1) = PERMDD || 'FILE UPDATE -- DELETION OF RECORDS';

25:    /* OPEN FILES */
26:    ON UNDEFINEDFILE (DATA) BEGIN;
27:       PRINTER = '*** ' || DDNAME || ' DD STATEMENT MISSING';
28:       CALL PRINTX (3);
29:       GOTO RETURN;
30:       END;
31:    OPEN FILE (DATA) INPUT RECORD TITLE (DDNAME);
32:    ON ENDFILE (DATA) GOTO CLOSE;
33:    OPEN FILE (PERM) UPDATE DIRECT TITLE (PERMDD);
34:    ON KEY (PERM) BEGIN;
35:       PRINTER = '*** RECORD DOES NOT EXIST IN FILE';
36:       CALL PRINTX (1);
37:       GOTO READ_DATA;
38:       END;


39: /***** MAIN EXECUTION LOOP *****/

40: READ_DATA:
41:    READ FILE (DATA) SET (PTR_DATA);
42:    PRINTER = '      ' || KEY;
43:    CALL PRINTX (2);
44:    DELETE FILE (PERM) KEY (KEY);
45:    GOTO READ_DATA;

46: CLOSE:
```

```
/* :UPDATE,FILE=ROADLOG,FUNCTION=DELETE,DDNAME=XXXXX */

 47:     CLOSE FILE (PERM);
 48:     CLOSE FILE (DATA);
 49:     CALL EXIT (PARM);

 50: RETURN:

 51: END DELETE;
```

FUNCTION=INSERT:

Member Name	PDRI	
Language	PL/I	
Subroutines	PRINTX1	
	RLGCON	
	SRTYPR (used by RLGCON)	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- UPDATE messages
	ROADLOG	-- Roadlog file
	PROJTBL	-- Project class table (RLGCON)
	SURFTBL	-- Surface type table (SRTYPR)
	any name	-- Roadlog data cards
Instruction	1 - 4	"PDRI"
	24 - 31	Name of input DD statement

Data cards, when inserting records, contain a complete Roadlog record. A two-card sequence is required for coding all Roadlog fields. Descriptor records, however, may be specified with a single card. The data card formats may be found in the publication Highway Information System Volume 1: User Information. The first card always contain a "1" in column 1 of a data card, and the second card contains a "2." Both card types contain the key in columns 2 through 14. The program reads a card, and checks for a "1." If not present, an error message is generated. If the card contains a "1," a second card is read. If this card either does not contain a "2," or contains a key other than that on the first, the program assumes that only one card has been supplied for that record. If the remark code specifies a mileage record (blank, "LP," "SP," "OS," or "NE"), an error message is generated, and the record not inserted. After inserting the record (or printing an error message if a record already exists with the specified key, raising the key condition), the program then returns to the beginning of the loop. If the second card read was not the "2" card for the preceding "1" card, it is used; otherwise, another card is read. The end-of-file condition may be raised when attempting to read a second card. If this occurs, the record is inserted (if a descriptor record), and the program terminates. Subroutine RLGCON performs the conversion

of the two-card sequence (the program sets up a dummy second card for descriptor records not having a second card) into the standard Roadlog file format.

The PDRI program listing follows:

```
INSERT: PROC(PARM) OPTIONS(MAIN);

 1: INSERT: PROC(PARM) OPTIONS(MAIN);
 2: /*  :UPDATE,FILE=ROADLOG,FUNCTION=INSERT,DDNAME=XXXXX */
 3:  DECLARE
 4:   INSTR CHAR(80) EXT,
 5:   DDNAME CHAR(8) DEF INSTR POS(24),
 6:   #_HDGS PIC'Z' DEF INSTR POS(72),
 7:   PARM CHAR(100),
 8:   (HEADING(9),PRINTER)CHAR(132) EXT,
 9:   PRINTXA ENTRY (PIC'Z',PIC'ZZ'),
10:   PRINTX ENTRY(PIC'Z'),
11:   RECORD CHAR(120),
12:   (CARD1,CARD2)CHAR(80),
13:   (FLAG,FLAG1) BIN FIXED,
14:   PERMDD CHAR(8) STATIC INIT('ROADLOG'),
15:   DATA FILE RECORD,
16:   PERM FILE RECORD KEYED ENV(INDEXED),
17:   CD CHAR(80) INIT(' '),
18:   CARD CHAR(80) BASED (PTR),
19:   TYPE_CODE CHAR(1),
20:   KEY CHAR (13),
21:   C1 (80) CHAR (1) DEF CARD1,
22:   COUNTY_# PIC'ZZ' DEF CARD1 POS(67),
23:   CITY_# PIC'ZZZ' DEF CARD2 POS(18),
24:   C2 (80) CHAR(1) DEF CARD2,
25:    C3 DEC FIXED(3,0),
26:   C4 CHAR(2);
27: /**INITIALIZATION**/
28:   CALL INIT (PARM);
29:     CALL RLGCON;
30:   #_HDGS=2;
31:   HEADING(1)=PERMDD||'FILE UPDATE--INSERTION OF RECORDS';
32:   ON UNDEFINEDFILE(DATA) BEGIN;
33:   PRINTER='**'||DDNAME||'DD STATEMENT MISSING';
34:    CALL PRINTX(3);
35:    GOTO FINISH;
36:    END;
37:    OPEN FILE(DATA) INPUT RECORD TITLE(DDNAME);
38:    ON ENDFILE(DATA) GOTO FINISH;
39:    OPEN FILE(PERM) UPDATE DIRECT TITLE(PERMDD);
40:    ON KEY(PERM) BEGIN;
41:     PRINTER='***ATTEMPT TO INSERT OVER EXISTING RECORD';
42:     CALL PRINTX(1);
43:    PRINTER=' ';
44:    CALL PRINTX(1);
45:     GOTO READ_DATA;
46:     END;
47:    FLAG=0;  FLAG1=0;
48: /* EXECUTION LOOP */
49: READ_DATA:
50:    IF FLAG1=1 THEN GOTO CLOSE;
51:    IF FLAG=0 THEN READ FILE(DATA) SET(PTR);
52:    FLAG=0;
53:    PRINTER='     '||CARD;
54:    CALL PRINTX (2);
55:     IF SUBSTR(CARD,1,1)¬='1' THEN DO;
56:     PRINTER='**ERROR IN CARD CODE, COLUMN 1, RECORD NOT INSERTED';
57:    CALL PRINTX(1);
58:    PRINTER=' ';
```

```
INSERT: PROC(PARM) OPTIONS(MAIN);

 59:     CALL PRINTX(1);
 60:     IF FLAG1=1 THEN GOTO CLOSE;
 61:      READ FILE(DATA) SET(PTR);
 62:      IF SUBSTR(CARD,1,1)='2' THEN GOTO READ_DATA;
 63:      FLAG=1;
 64:      GOTO READ_DATA;
 65:       END;
 66:     IF SUBSTR(CARD,9,1)='+' THEN DO;
 67:       TYPE_CODE='L';
 68:       KEY=SUBSTR(CARD,2,13);
 69:       CARD1=' '||SUBSTR(CARD,2);
 70:      END;
 71:      ELSE DO;
 72:       TYPE_CODE='S';
 73:       KEY=SUBSTR(CARD,2,9);
 74:       CARD1=' '||SUBSTR(CARD,2,7)||'+0.'||SUBSTR(CARD,9,2)||'0'||
 75:         SUBSTR(CARD,11,66);
 76:       END;
 77:     READ FILE(DATA) SET(PTR);
 78:     IF FLAG1=1 THEN GOTO CLOSE;
 79:     IF SUBSTR(CARD,1,1)¬='2' THEN GOTO DS;
 80:    IF TYPE_CODE='L' & SUBSTR(CARD,2,13)¬=KEY|TYPE_CODE='S'& SUBSTR
 81:       (CARD,2,9)¬= KEY THEN DO;
 82:        C4=SUBSTR(CARD1,65,2);
 83:        IF C4=' '|C4='NE'|C4='SP'|C4='LP'|C4='OS' THEN DO;
 84:        PRINTER='**2ND CARD MISSING, REQUIRED FOR MILEAGE RECORD,'||
 85:                ' RECORD NOT INSERTED';
 86:     CALL PRINTX(1);
 87:     PRINTER=' ';
 88:     CALL PRINTX(1);
 89:        FLAG=3;
 90:        GOTO READ_DATA;
 91:          END;
 92:        ELSE DO;
 93:        FLAG=3;
 94:        GOTO CNV1;
 95:        END;
 96:       END;
 97: S2:
 98:     PRINTER='     '||CARD;
 99:      CALL PRINTX(1);
100:     IF TYPE_CODE='L' THEN CARD2=' '||SUBSTR(CARD1,2,13)||SUBSTR(CARD,
101:        15);
102:      ELSE CARD2=' '||SUBSTR(CARD1,2,13)||SUBSTR(CARD,11);
103:      FLAG=0;
104:        GOTO CON1;
105: DS:    C4=SUBSTR(CARD1,65,2);
106:   IF C4=' '|C4='NE'|C4='LP'|C4='OS'|C4='SP' THEN GOTO ESC;
107:     GOTO CNV1;
108: ESC:
109:     PRINTER='     '||CARD;
110:     CALL PRINTX(1);
111:     PRINTER='**CARD INCORRECT, EITHER INCORRECT KEY, DUPLICATE CARD OR'
112:       ||' OUT OF SEQUENCE**';
113:     CALL PRINTX(1);
114:     FLAG=1;
115:      GOTO READ_DATA;
116: CNV1: CARD2=' '||SUBSTR(CARD1,2,13)||SUBSTR(CD,14);
```

```
INSERT: PROC(PARM) OPTIONS(MAIN);

117: CON1:
118:    C4=SUBSTR(CARD1,65,2);
119:    IF C4='DS'|C4='ER'|C4='EN'|C4='CO'|C4='IL' THEN GOTO INST;
120:    DO I=3 TO 8,10,12 TO 14,50 TO 53,67 TO 72;
121:      IF(C1(I)<'0'|C1(I)>'9')&C1(I)¬=' ' THEN DO;
122:        PRINTER='***ERROR IN CODING CARD COLUMN: '||I||' ON CARD1'||
123:                ' SHOULD BE NUMERIC OR BLANK, RECORD NOT INSERTED';
124:     CALL PRINTX(1);
125:     PRINTER=' ';
126:     CALL PRINTX(1);
127:        GOTO READ_DATA;
128:        END;
129:    END;
130:    DO I=15,17 TO 66;
131:      IF(C2(I)<'0'|C2(I)>'9')&C2(I)¬=' ' THEN DO;
132:        PRINTER='***ERROR IN CODING CARD COLUMN: '||I||' ON CARD2'||
133:                ' SHOULD BE NUMERIC OR BLANK, RECORD NOT INSERTED';
134:     CALL PRINTX(1);
135:     PRINTER=' ';
136:     CALL PRINTX(1);
137:        GOTO READ_DATA;
138:        END;
139:    END;
140:    IF C1(2)¬='I' & C1(2)¬='P' & C1(2)¬='S' THEN DO;
141:      PRINTER='***ERROR IN ROUTE SYSTEM CODE - RECORD NOT INSERTED';
142:      CALL PRINTX(1);
143:     PRINTER=' ';
144:      GOTO READ_DATA;
145:      END;
146:     IF COUNTY_#>56 THEN DO;
147:        PRINTER='***ERROR IN COUNTY_# CODE - RECORD NOT INSERTED';
148:      CALL PRINTX(1);
149:     PRINTER=' ';
150:     CALL PRINTX(1);
151:      GOTO READ_DATA;
152:      END;
153:     IF CITY_#>126 THEN DO;
154:       PRINTER='***ERROR IN CITY NUMBER CODE - RECORD NOT INSERTED';
155:     CALL PRINTX(1);
156:     PRINTER=' ';
157:     CALL PRINTX(1);
158:       GOTO READ_DATA;
159:       END;
160: INST: CALL COV1(CARD1,CARD2,RECORD);
161:     WRITE FILE(PERM) FROM(RECORD) KEYFROM(SUBSTR(RECORD,2,13));
162:     GOTO READ_DATA;
163: FINISH:
164:     IF SUBSTR(CARD,1,1)¬='1' | FLAG1=1 THEN GOTO CLOSE;
165:     C4=SUBSTR(CARD,65,2);
166:    IF C4¬=' '&C4¬='NE'&C4¬='SP'&C4¬='LP'&C4¬='OS' THEN DO;
167:     FLAG1=1;
168:      GOTO CNV1;
169:     END;
170:     ELSE DO;
171:     FLAG=1;
172:      FLAG1=1;
173:     GOTO ESC;
174:  CLOSE: PRINTER='****END OF DATA ****'    ;
```

```
INSERT: PROC(PARM) OPTIONS(MAIN);

175:        CALL PRINTX(3);
176:        CLOSE FILE(PERM);
177:        CLOSE FILE(DATA);
178:        CALL EXIT (PARM);
179:    END INSERT;
```

FUNCTION=REWRITE:

Member Name	PDRR
Language	PL/I
Subroutines	PRINTX1 RLGCON SRTYPR (used by RLGCON)
Files	SYSPRINT -- IBM messages PRINTER -- UPDATE messages ROADLOG -- Roadlog file PROJTBL -- Project class table (RLGCON) SURFTBL -- Surface type table (SRTYPR) any name -- Roadlog data cards
Instruction	1 - 4 "PDRR" 24 - 31 Name of input DD statement

Data cards for rewriting, though in the same format as those for inserting, require only fields being altered to be coded. Hence, a "1" card, a "2" card, or both may be supplied. Because of the low volume of updates performed on the Roadlog file, the program operates with one card at a time. Hence, if both cards are included, two separate rewrites will be performed. This method of operation greatly simplifies the program logic. The program reads one data card. If the card is a "1" card, a dummy "2" card is set up; if it is a "2" card, a dummy "1" card is set up. The Roadlog record is then read, and RLGCON used to convert it into character format. The record, in character format, is then compared with the data cards to determine which fields are being altered. After performing the alterations, RLGCON is again used to return to Roadlog file format, and the record rewritten.

The PDRR program listing follows:

```
/* :UPDATE, FILE=ROADLOG,FUNCTION=REWRITE,DDNAME=XXXXX */

  1: /* :UPDATE, FILE=ROADLOG,FUNCTION=REWRITE,DDNAME=XXXXX */
  2: REWRITE: PROC(PARM) OPTIONS(MAIN);
  3: /* DECLARATION FOR INSTRUCTION & PRINT ROUTINE */
  4: DECLARE
  5:  PARM CHAR(100),
  6:  INSTR CHAR(80) EXT,
  7:  #_HDGS PIC'Z' DEF INSTR POS(72),
  8:  COUNTY_# PIC'ZZ' DEF CARD1 POS(67),
  9:  CITY_# PIC'ZZZ' DEF CARD2 POS(18),
 10:  DDNAME CHAR(8) DEF INSTR POS(24),
 11:  (PRINTER,HEADING(9)) CHAR(132) EXT,
 12:  PRINTX ENTRY(PIC'Z');
 13: /* DECLARATION FOR DATA INPUT & ROADLOG FILE */
 14: DECLARE
 15:  RECORD CHAR(120),
 16:  RE(120) CHAR(1) DEF RECORD,
 17:  (CARD1,CARD2) CHAR(80),
 18:  C1 (80) CHAR(1) DEF CARD1,
 19:   C2 (80) CHAR(1) DEF CARD2,
 20:   CARD CHAR(80) BASED(PTR),
 21:   C3(80) CHAR(1) BASED(PTR),
 22:   C4 DEC FIXED(3,0),
 23:  NEWREC CHAR(132),
 24:  NR(132) CHAR(1) DEF NEWREC,
 25:  DATA FILE RECORD SEQL INPUT,
 26:  ROADLOG FILE RECORD DIRECT UPDATE KEYED ENV(INDEXED);
 27: /* OTHER VARIABLES */
 28: DECLARE
 29:  KEY CHAR(13),
 30:   FLAG BIN FIXED;
 31: /***INITIALIZATION***/
 32:  CALL INIT(PARM);
 33:  CALL RLGCON;
 34:  #_HDGS = 2;
 35:  HEADING(1)='     ROADLOG -- FUNCTION=REWRITE';
 36:  OPEN
 37:     FILE(DATA)TITLE(DDNAME),
 38:     FILE (ROADLOG);
 39:  ON KEY (ROADLOG)BEGIN;
 40:    PRINTER='** NO RECORD FOR ATTEMPTED REWRITE';
 41:     CALL PRINTX(1);
 42:     GOTO READ_DATA;
 43:     END;
 44:   ON ENDFILE (DATA) GO TO FINISH;
 45: /* EXECUTION */
 46: READ_DATA:
 47:  READ FILE (DATA) SET(PTR);
 48:     PRINTER='  '||CARD;
 49:     CALL PRINTX(2);
 50:     IF C3(1)='1' THEN GOTO CARD_1;
 51:     IF C3(1)='2' THEN GOTO CARD_2;
 52:     PRINTER='** INCORRECT CARD CODE IN COLUMN 1';
 53:     CALL PRINTX(2);
 54:     GOTO READ_DATA;
 55: CARD_1:
 56:  IF SUBSTR(CARD,9,1)='+' THEN DO;
 57:     NEWREC=' '||SUBSTR(CARD,2);
 58:     KEY=SUBSTR(CARD,2,13);
```

```
/* :UPDATE, FILE=ROADLOG,FUNCTION=REWRITE,DDNAME=XXXXX */

 59:      END;
 60:   ELSE DO;
 61:      NEWREC=' '||SUBSTR(CARD,2,7)||'+0.'||SUBSTR(CARD,9,2)||'0'||
 62:        SUBSTR(CARD,11,66);
 63:     KEY=SUBSTR(NEWREC,2,13);
 64:      END;
 65:    GOTO CN2;
 66: CARD_2:
 67:     IF SUBSTR(CARD,9,1)='+' THEN DO;
 68:      SUBSTR(NEWREC,81)=SUBSTR(CARD,15);
 69:      SUBSTR(NEWREC,2,13)=SUBSTR(CARD,2,13);
 70:      END;
 71:     ELSE DO;
 72:      NEWREC=' '||SUBSTR(CARD,2,7)||'+.0'||SUBSTR(CARD,9,2)||'0'||' ';
 73:      SUBSTR(NEWREC,81)=SUBSTR(CARD,11);
 74:      END;
 75:     KEY=SUBSTR(NEWREC,2,13);
 76:     CARD2=' '||SUBSTR(NEWREC,2,13)||SUBSTR(NEWREC,81);
 77: CN2:
 78:     CARD1=' '||KEY||SUBSTR(NEWREC,15,66);
 79:     CARD2=' '||KEY||SUBSTR(NEWREC,81);
 80:    IF C1(2)¬='I' & C1(2)¬='P' & C1(2)¬='S' THEN DO;
 81:       PRINTER='***ERROR IN ROUTE SYSTEM CODE IN FILE RECORD OR' ||
 82:               ' IN NEW REWRITE DATA***';
 83:      CALL PRINTX(2);
 84:      GOTO READ_DATA;
 85:      END;
 86:     IF COUNTY_#>56 THEN DO;
 87:      PRINTER='***ERROR IN COUNTY NUMBER CODE IN NEW REWRITE DATA**';
 88:       CALL PRINTX(2);
 89:      GOTO READ_DATA;
 90:      END;
 91:     IF CITY_#>126 THEN DO;
 92:      PRINTER='***ERROR IN CITY NUMBER CODE IN NEW REWRITE DATA**';
 93:       CALL PRINTX(2);
 94:      GOTO READ_DATA;
 95:      END;
 96:    READ FILE(ROADLOG) INTO(RECORD) KEY(SUBSTR(NEWREC,2,13));
 97:    CALL COV2 (CARD1,CARD2,RECORD);
 98:     DO I=50 TO 53,65 TO 132;
 99:      IF NR(I)¬=' ' THEN DO;
100:     IF NR(I)='$' THEN NR(I)=' ';
101:     IF I<81 THEN C1(I)=NR(I);
102:    ELSE C2(I-66)=NR(I);
103:       END;
104:     END;
105:   DO I=3 TO 8,10,12 TO 14,50 TO 53,67 TO 72;
106:    IF(C1(I)<'0'|C1(I)>'9')& C1(I)¬=' ' THEN GO TO ERR;
107:    END;
108:      DO I=15,17 TO 66;
109:    IF(C2(I)<'0'|C2(I)>'9')& C2(I)¬=' ' THEN GOTO ERR;
110:    END;
111:     DO I=15 TO 49;
112:      IF NR(I)¬=' ' THEN GOTO CN4;
113:      END;
114:      GOTO CN3;
115:   CN4: SUBSTR(CARD1,15,35)=SUBSTR(NEWREC,15,35);
116:   CN3: DO I=54 TO 64;
```

```
/* :UPDATE, FILE=ROADLOG,FUNCTION=REWRITE,DDNAME=XXXXX */

117:      IF NR(I)¬=' ' THEN GOTO CN5;
118:      END;
119:      GOTO CN6;
120:   CN5: SUBSTR(CARD1,54,11)=SUBSTR(NEWREC,54,11);
121:  CN6: DO I=15 TO 49,54 TO 64;
122:        IF C1(I)='$' THEN C1(I)=' ';
123:      END;
124:      CALL COV1(CARD1,CARD2,RECORD);
125:    REWRITE FILE(ROADLOG)FROM (RECORD) KEY(SUBSTR(RECORD,2));
126:     GOTO READ_DATA;
127:   ERR:
128:   PRINTER='**ERROR IN CODING OF NUMERIC FIELDS, RECORD NOT REWRITTEN';
129:    CALL PRINTX(2);
130:    GOTO READ_DATA;
131: FINISH:
132:     PRINTER='   END OF DATA';
133:     CALL PRINTX(3);
134:     CLOSE
135:       FILE(ROADLOG),
136:       FILE(DATA);
137:     CALL EXIT(PARM);
138:     END REWRITE;
```

FUNCTION=NEW-KEY:

Member Name	PDRN	
Language	PL/I	
Subroutines	PRINTX1	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- UPDATE messages
	ROADLOG	-- Roadlog file
	any name	-- Roadlog data cards
Instruction	1 - 4	"PDRN"
	24 - 31	Name of input DD statement

The NEW-KEY function allows alteration of the key field, which cannot be performed by REWRITE. Upon reading a data card (containing the existing key in columns 1-13, and the new key in columns 15-27), the program first checks to be sure that no record already exists with the new key, then reads the existing record, supplies the new key, inserts the record, and deletes the old record. An error message is generated either if a record already exists with the new key, or if no record exists with the old key.

The PDRN program listing follows:

```
/* :UPDATE,FILE=ROADLOG,FUNCTION=NEW_KEY,DDNAME=XXXXX */

 1: /* :UPDATE,FILE=ROADLOG,FUNCTION=NEW_KEY,DDNAME=XXXXX */

 2: NEWKEY:  PROCEDURE (PARM) OPTIONS (MAIN);

 3: /* INSTRUCTION */
 4: DECLARE
 5:    INSTR CHAR(80) EXT,
 6:    DDNAME CHAR(8) DEF INSTR POS(24),
 7:    #_HDGS PIC'Z' DEF INSTR POS(72);

 8: /* PRINT ROUTINE */
 9: DECLARE
10:    PARM CHAR(100),
11:   (HEADING(9),PRINTER) CHAR(132) EXT,
12:    PRINTX ENTRY (PIC'Z'),
13:    PRINTXA ENTRY (PIC'Z',PIC'ZZ');

14: /* PERMANENT FILE */
15: DECLARE
16:     RECORD CHAR(120) STATIC,
17:      1 R DEF RECORD,
18:        3  DUM1 CHAR(1),
19:        3  KEY CHAR(13),
20:    PERMDD CHAR(8) STATIC INIT ('ROADLOG'),
21:    PERM FILE RECORD KEYED ENV (INDEXED);

22: /* DATA INPUT */
23: DECLARE
24:    CARD CHAR(80) BASED (PTR_DATA),
25:    1  C BASED (PTR_DATA),
26:       3  OLD CHAR(13),
27:       3  DUM CHAR(1),
28:       3  NEW CHAR(13),
29:    DATA FILE RECORD;


30: /***** INITIALIZATION *****/

31:    CALL INIT (PARM);
32:    #_HDGS = 2;
33:    HEADING(1) = PERMDD || 'FILE UPDATE -- NEW KEY';

34:    /* INIT FILES */
35:    ON UNDEFINEDFILE (DATA) BEGIN;
36:       PRINTER = '*** ' || DDNAME || ' DD STATEMENT MISSING';
37:       CALL PRINTX (3);
38:       GOTO RETURN;
39:       END;
40:    OPEN FILE (DATA) INPUT RECORD TITLE (DDNAME);
41:    OPEN FILE (PERM) UPDATE DIRECT TITLE (PERMDD);
42:    ON ENDFILE (DATA) GOTO FINISH;


43: /***** MAIN EXECUTION LOOP *****/

44: READ_DATA:
45:    READ FILE (DATA) SET (PTR_DATA);
46:    PRINTER = '      ' || CARD;
```

```
/* :UPDATE,FILE=ROADLOG,FUNCTION=NEW_KEY,DDNAME=XXXXX */

47:     CALL PRINTXA (3,7);
48:     ON KEY (PERM) GOTO GET_RECORD;
49:      READ FILE(PERM) INTO(RECORD) KEY(C.NEW);
50:     PRINTER ='*** ATTEMPT TO INSERT OVER EXISTING RECORD';
51:     CALL PRINTX (1);
52:     GOTO READ_DATA;

53: GET_RECORD:
54:     ON KEY (PERM) BEGIN;
55:        PRINTER = '*** RECORD DOES NOT EXIST IN FILE';
56:        CALL PRINTX (1);
57:        GOTO READ_DATA;
58:        END;
59:      READ FILE(PERM) INTO(RECORD) KEY(C.OLD);
60:     R.KEY = C.NEW;
61:     WRITE FILE (PERM) FROM (RECORD) KEYFROM (R.KEY);
62:     DELETE FILE(PERM) KEY(C.OLD);
63:     GOTO READ_DATA;

64: FINISH:
65:     PRINTER = '  END OF DATA';
66:     CALL PRINTX (3);
67:     CLOSE FILE (PERM);
68:     CLOSE FILE (DATA);

69: RETURN:
70:     CALL EXIT (PARM);

71: END NEWKEY;
```

COPY --

Member Name	PBR	
Language	PL/I	
Subroutines	PRINTX1	
	RLGCON	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- COPY printer output
	ROADLOG	-- Roadlog file
	SAVERLG	-- Backup copy (output)
Instruction	1 - 3	"PBR"
	5	"Y"/"N" for LIST=YES/LIST=NO

COPY prepares a backup copy of the Roadlog file. The backup copy is a sequential version of the Roadlog file, with identical record length (120 characters). A dummy record containing the date is first written. This record is followed by the Roadlog records. If LIST=YES is specified, RLGCON is used to convert the Roadlog records into character format, and the records are listed in the same "dump" format as when DUMP is used to list the file. A count is taken of the number of records in the file. The count is printed after the last record is written.

The PBR program listing follows:

```
/* :COPY,FILE=ROADLOG,LIST=YES/NO */

 1: /* :COPY,FILE=ROADLOG,LIST=YES/NO */

 2: COPY:  PROCEDURE (PARM) OPTIONS (MAIN);

 3: /* INSTRUCTION */
 4: DECLARE
 5:    INSTR CHAR(80) EXT,
 6:    LIST CHAR(1) DEF INSTR POS(5),
 7:    #_HDGS PIC'Z' DEF INSTR POS(72);

 8: /* PRINT ROUTINE */
 9: DECLARE
10:    PARM CHAR(100),
11:   (HEADING(9),PRINTER) CHAR(132) EXT,
12:    PRINTX ENTRY (PIC'Z');

13: /* FILES */
14: DECLARE
15:    (CARD1,CARD2) CHAR(80),
16:     RECORD CHAR(120) BASED(PTR),
17:    BACKDD CHAR(8) STATIC INIT ('SAVERLG'),
18:    PERMDD CHAR(8) STATIC INIT ('ROADLOG'),
19:    PERM FILE RECORD KEYED ENV (INDEXED),
20:    BACKUP FILE RECORD;

21: /* OTHER VARIABLES */
22: DECLARE
23:    UD CHAR(6),
24:    CNTR BIN FIXED (31),
25:    PCNTR PIC'ZZZZZ9';


26: /***** INITIALIZATION *****/

27:    CALL INIT (PARM);

28:    /* SET UP HEADINGS */
29:    #_HDGS = 2;
30:    HEADING(1) = PERMDD || 'FILE COPY ROUTINE';

31:    /* INIT FILES */
32:    OPEN FILE (PERM) INPUT TITLE (PERMDD);
33:    OPEN FILE (BACKUP) OUTPUT TITLE (BACKDD);
34:    ON ENDFILE (PERM) GOTO DONE;

35:    /* RECORD DATE */
36:    UD = DATE;
37:    PTR = ADDR(HEADING(9));
38:    RECORD = SUBSTR(UD,3,2) || '/' ||
39:         SUBSTR(UD,5,2) || '/' || SUBSTR(UD,1,2);
40:    WRITE FILE (BACKUP) FROM (RECORD);


41: /***** MAIN EXECUTION LOOP *****/

42:    DO CNTR=1 TO 999999;
43:       READ FILE (PERM) SET (PTR);
44:       WRITE FILE (BACKUP) FROM (RECORD);
```

```
/* :COPY,FILE=ROADLOG,LIST=YES/NO */

 45:        IF LIST='Y' THEN DO;
 46:      PRINTER=SUBSTR(CARD1,2)||SUBSTR(CARD2,15);
 47:     PRINTER=CARD1||SUBSTR(CARD2,15);
 48:           CALL PRINTX (1);
 49:           END;
 50:        END;

 51: DONE:
 52:     PCNTR = CNTR - 1;
 53:     PRINTER = 'NUMBER OF RECORDS IN FILE:  ' || PCNTR;
 54:     CALL PRINTX (3);
 55:     CLOSE FILE (PERM);
 56:     CLOSE FILE (BACKUP);
 57:     CALL EXIT (PARM);

 58: END COPY;
```

CREATE --

Member Name	PAR	
Language	PL/I	
Subroutines	PRINTX1	
	RLGCON	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- CREATE output
	ROADLOG	-- Roadlog file (output)
	SAVERLG	-- Backup copy
Instruction	1 - 3	"PAR"
	5	"Y"/"N" for LIST=YES/LIST=NO

CREATE restores the Roadlog file from a backup copy saved via program COPY. The first record in the file is a dummy record, containing the date on which the file was copied. This date is printed prior to performing the create operation. After printing the date, the records are read from the backup copy and written into the Roadlog file, destroying the previous file. If LIST=YES is specified, subroutine RLGCON is used to convert the records to character format for printing. As with COPY, the records are counted as they are written. The count is printed after the create operation is complete.

The PAR program listing follows:

```
/* :CREATE,FILE=ROADLOG,LIST=YES/NO */

 1: /* :CREATE,FILE=ROADLOG,LIST=YES/NO */

 2: CREATE:  PROCEDURE (PARM) OPTIONS (MAIN);

 3: /* INSTRUCTION */
 4: DECLARE
 5:    INSTR CHAR(80) EXT,
 6:    LIST CHAR(1) DEF INSTR POS(5),
 7:    #_HDGS PIC'Z' DEF INSTR POS(72);

 8: /* PRINT ROUTINE */
 9: DECLARE
10:    PARM CHAR(100),
11:   (HEADING(9),PRINTER) CHAR(132) EXT,
12:    PRINTX ENTRY (PIC'Z');

13: /* FILES */
14: DECLARE
15:    (CARD1,CARD2)CHAR(80),
16:    RECORD CHAR(120) BASED(PTR),
17:    BACKDD CHAR(8) STATIC INIT ('SAVERLG'),
18:    PERMDD CHAR(8) STATIC INIT ('ROADLOG'),
19:    PERM FILE RECORD KEYED ENV (INDEXED),
20:    BACKUP FILE RECORD;

21: /* OTHER VARIABLES */
22: DECLARE
23:    CNTR BIN FIXED (31),
24:    PCNTR PIC'ZZZZZ9';


25: /***** INITIALIZATION *****/

26:    CALL INIT (PARM);

27:    /* SET UP HEADINGS */
28:    #_HDGS = 2;
29:    HEADING(1) = PERMDD || 'FILE CREATION ROUTINE';

30:    /* INIT FILES */
31:    OPEN FILE (BACKUP) INPUT TITLE (BACKDD);
32:    OPEN FILE (PERM) OUTPUT TITLE (PERMDD);
33:    ON ENDFILE (BACKUP) GOTO DONE;

34:    /* PRINT DATE */
35:    READ FILE (BACKUP) SET (PTR);
36:    PRINTER = '   DATE OF BACKUP FILE IS ' || RECORD;
37:    CALL PRINTX (1);
38:    PRINTER = ' ';
39:    CALL PRINTX (1);


40: /***** MAIN EXECUTION LOOP *****/

41:    DO CNTR=1 TO 999999;
42:       READ FILE (BACKUP) SET (PTR);
43:       WRITE FILE (PERM) FROM (RECORD) KEYFROM (SUBSTR(RECORD,2));
44:       IF LIST='Y' THEN DO;
```

```
/* :CREATE,FILE=ROADLOG,LIST=YES/NO */

 45:      CALL COV2(CARD1,CARD2,RECORD);
 46:      PRINTER=CARD1||SUBSTR(CARD2,15);
 47:      PRINTER=SUBSTR(CARD1,2)||SUBSTR(CARD2,15);
 48:           END;
 49:        END;

 50: DONE:
 51:     PCNTR = CNTR - 1;
 52:     PRINTER = 'NUMBER OF RECORDS IN FILE:  ' || PCNTR;
 53:     CALL PRINTX (3);
 54:     CLOSE FILE (PERM);
 55:     CLOSE FILE (BACKUP);
 56:     CALL EXIT (PARM);

 57: END CREATE;
```

LIST-&-SUM --

Member Name	NAR	
Language	PL/I	
Subroutines	PRINTX1	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- LIST-&-SUM output
	ROADLOG	-- Roadlog file
	CNTYTBL	-- Table of county names
Instruction	1 - 3	"NAR"
	40 - 43	Beginning route number
	56 - 59	Ending route number

LIST-&-SUM provides the main listing of Roadlog data that forms the bulk of the annual Roadlog report. It always operates on entire routes; if the user specifies a portion of a route on his command, the entire route is processed. For each route processed in a run, two steps are performed. The first step is to read the records for the route, list the records in the proper format, and save the mileages in arrays for printing by county and by location code. The second step, performed after the "EN" record indicating the end of the route has been printed, is to print the county summary. LIST-&-SUM processes only those Roadlog records with remark codes " ," "SP," "LP," "OS," "NE," "DS," "ER," "EN," and "CO." All other records are by-passed. "DS" records are descriptive records. The description field is printed, centered on the page. A blank line precedes and follows the description. This record type is used for indicating signed route numbers and the beginnings of spurs and loops. "ER" records, like "DS" records, contain only a description. "ER" description, however, are not centered and are not set off with blank lines. "EN" records indicate the ends of routes. Each time an "EN" record is encountered, the following tasks are performed: 1) the "EN" description and milepoint are printed, 2) the county summary is printed, and 3) variables are re-initialized for processing the next route. "CO" records indicate coincident sections. When one of these is encountered, the milepoint and description fields are printed. The starting and ending keys of the coincident section are retrieved from the description, and the coincident section scanned. The section lengths in the coincident section are added to the accumulated length, which is then printed on the same line as the description and milepoint. The

coincident mileage is not included in the county summary. The other five record types are "mileage" records. When a mileage record is read, it is first formatted for printing and printed. The mileage is then stored in an array for future use. The summary by counties shows the mileage broken down by the 56 counties, and by location. The nine location breakdowns are: route length, constructed length, unimproved length, wye length, municipal, county, national forest, Indian reservation, and other. Three further classifications are made: spurs, loops, and all others. The spurs and loops are shown separately. Setting up a directly-addressed array for the summary would require a 56x9x3 array of decimal (7,3) variables, or 6048 bytes of storage. In order to allow operation in a small partition, an indirect-addressing technique is used in the program. The indirect method reduces the amount of core required to 25x9 decimal variables, or 900 bytes (plus a small amount of overhead core). MBC is an array of 25 structures, each having 9 decimal variables for the storage of one county. ACNTY is a 56x3 array of binary variables, which performs the addressing into MBC. At the beginning of each route, both MBC and ACNTY are set to zero. An additional variable, #_COUNTIES, keeps track of the number of elements of MBC utilized (also initialized to zero). The first record of a route is never a spur or loop. The second element of ACNTY has a value of 2 for "SP" (spur) records, 3 for "LP" (loop) records, and 1 for all other mileage records. Hence, this will have a value of 1 for the first mileage record. The first subscript of ACNTY is the county number. The county number in the Roadlog record is retrieved, and use to access ACNTY. The element of ACNTY, found to be zero, is set to 1 (indicating that the first structure of MBC will be used), and #_COUNTIES is incremented. Whenever a succeeding record contains the same county number, and falls into the same spur/loop/other category, the program will find that the element of ACNTY accessed is already non-zero, and will use the structure of MBC pointed to by the ACNTY element. Whenever the element of ACNTY accessed is zero, another structure of MBC is allocated, #_COUNTIES incremented, and the element of ACNTY altered to indicate the number of the structure used. MBC presently is allocated 25 structures, which is well more than required for any present routes. If, during the processing of a route, a 26th structure is required, the program will print the message:

*** INSUFFICIENT STORAGE FOR COUNTY SUMMARY

Should this occur, MBC must be allocated more structures, and the test (in the SAVE_MILEAGES portion of the program) for MBC overflow altered to indicate the new size.

The NAR program listing follows:

```
/* :LIST_&_SUM,REPORT=ROADLOG,DATA=XXXXX */

  1: /* :LIST_&_SUM,REPORT=ROADLOG,DATA=XXXXX */

  2: LISTSUM:  PROCEDURE (PARM) OPTIONS (MAIN);


  3: /***** DECLARATION OF VARIABLES (ALPHABETICAL ORDER) *****/

  4: DECLARE
  5:    #_COUNTIES DEC FIXED (3,0) STATIC,
  6:    #_HDGS PIC'Z' DEF INSTR POS(72),
  7:    #_LINES PIC'ZZ' STATIC,
  8:    ACCUM_MILEAGE DEC FIXED (7,3) STATIC,
  9:    ACNTY(56,3) BIN FIXED STATIC,
 10:    C2 CHAR(2),
 11:    CODE(3) CHAR(7) INIT(' ',' (SPUR)',' (LOOP)'),
 12:    COUNTY(56) CHAR(15) STATIC,
 13:    DFRLG CHAR(132) STATIC,
 14:    DUPLICATE DEC FIXED (5,3) STATIC,
 15:    ENDKEY CHAR(4) DEF INSTR POS(56),
 16:    F(0:9) PIC'Z' STATIC INIT (0,1,2,3,4,5,6,7,8,9),
 17:    1  FRLG DEF DFRLG,
 18:       3  DESCR CHAR(35),
 19:       3  MP,
 20:          5  MILEPOST PIC'ZZZZ9',
 21:          5  OFFSET CHAR(7),
 22:       3  MAINT_SEC PIC'ZZZZB',
 23:       3  PROJ_CLASS CHAR(12),
 24:       3  YR_BLT PIC'ZZBB',
 25:       3  YR_IMP PIC'ZZBBB',
 26:       3  SURTYP CHAR(4),
 27:       3  SURF_THICKNESS PIC'ZZZV.Z',
 28:       3  BASE_THICKNESS PIC'ZZZV.Z',
 29:       3  SURF_WIDTH PIC'ZZZZ',
 30:       3  RDWAY_WIDTH PIC'ZZZZZ',
 31:       3  #_LANES PIC'ZZZ',
 32:       3  SECTN PIC'ZZZZV.ZZZBB',
 33:       3  LOCATION CHAR(6),
 34:       3  CONST PIC'ZZZZV.ZZZ',
 35:       3  ACCUM_MLGE PIC'ZZZZZV.ZZZ',
 36:    HEADING(9) CHAR(132) EXT,
 37:    INIT_FLAG CHAR(1),
 38:    INSTR CHAR(80) EXT,
 39:   (KEY1,KEY2) CHAR(16),
 40:    LOC2(0:10) CHAR(6) STATIC INIT
 41:       ('******',' CITY ','COUNTY','NATFOR','INDRES',' GAME ',
 42:       'MILRES','NATMON','NATPRK','ST FOR','ST PRK'),
 43:    LINES CHAR(132) STATIC INIT ((132)'-'),
 44:    1  MBC(25) LIKE MLGE STATIC,
 45:    1  ML STATIC,
 46:       3  ROUTE PIC'ZZZV.ZZZ',
 47:       3  CONST PIC'ZZZZV.ZZZ',
 48:       3  UNIMP PIC'ZZZZV.ZZZ',
 49:       3  WYE PIC'ZZV.ZZZ',
 50:       3  CITY PIC'ZZZZV.ZZZ',
 51:       3  CNTY PIC'ZZZZV.ZZZ',
 52:       3  NFOR PIC'ZZZZV.ZZZ',
 53:       3  IRES PIC'ZZZZV.ZZZ',
 54:       3  OTHR PIC'ZZZV.ZZZ',
```

```
/* :LIST_&_SUM,REPORT=ROADLOG,DATA=XXXXX */

 55:     1  MLGE STATIC,
 56:        3 (ROUTE,CONST,UNIMP,WYE,CITY,CNTY,NFOR,IRES,OTHR)
 57:           DEC FIXED (7,3),
 58:     OUT_OF_STATE LIKE MLGE,
 59:     PAGE_POSITION PIC'ZZ' DEF INSTR POS(9),
 60:     PAGE_SIZE PIC'ZZ' DEF INSTR POS(7),
 61:     RECORD CHAR(80) BASED (PTR_TBL),
 62:     PARM CHAR(100),
 63:     PRINTER CHAR(132) EXT,  /** DATA FOR PRINTING **/
 64:     1  RLG BASED (PTR_RLG),
 65:        3  DUMMY1 CHAR(1),
 66:        3  KEY,
 67:           5  SYSTEM CHAR(1),
 68:           5  RT_# PIC'999',
 69:           5  MP,
 70:              7  MILEPOST PIC'ZZZ',
 71:           7 OFFSET CHAR(6),
 72:        3  REMARK CHAR(2),
 73:        3  (SECTN,ROUTE,CONST,UNIMP) DEC FIXED(5,3),
 74:        3  WYE DEC FIXED(3,3),
 75:        3  DESCR CHAR(35),
 76:        3  PROJ_CLASS CHAR(11),
 77:        3  DIVIDED_CODE CHAR(1),
 78:        3  (#_LANES,POPULATION_CODE)DEC FIXED(1,0),
 79:        3  (CITY_#,COUNTY_#,YR_BLT,YR_IMP,FORHWY_#,ADMIN_CODE,
 80:           LOCN(2),DUMM,SURF_WIDTH,RDWAY_WIDTH)DEC FIXED(3,0),
 81:        3  (SURF_THICKNESS,BASE_THICKNESS)DEC FIXED(3,1),
 82:        3  SURF_TYPE_CODE DEC FIXED(1,0),
 83:        3  (SURF_TYPE,MAINT_SEC) DEC FIXED(5,0),
 84:        3  DATE,
 85:           4 (MONTH,DAY,YEAR) DEC FIXED(3,0),
 86:        3 DUMMY2 CHAR(2),
 87:     1  RLG2 BASED (PTR_RLG),
 88:        3  DUM CHAR(1),
 89:        3  KEY CHAR(13),
 90:           3 DUMM1 CHAR(16),
 91:        3  CN,
 92:           5  DUM1 CHAR(5),
 93:           5  RT_# CHAR(4),
 94:           5  DUM2 CHAR(6),
 95:           5  KEY(2, CHAR(10),
 96:     ROADLOG FILE INT RECORD KEYED ENV (INDEXED GENKEY),
 97:     RURAL_MILEAGE PIC'ZZZZZV.ZZZ',
 98:     SAVE_KEY CHAR(16),
 99:     STARTKEY CHAR(4) DEF INSTR POS(40),
100:     STRING_ML CHAR(68) DEF ML,
101:     SURF(0:8) CHAR(3) STATIC INIT
102:        (' ','PRM','BLD','GRD','GRV','BST','RMS','PMS','PCC'),
103:     TOT_CONST PIC'ZZZZV.ZZZ',
104:     1  TOTALS LIKE MLGE STATIC,
105:     ZRT_# PIC'ZZZ';


106: /***** PROGRAM INITIALIZATION *****/

107:     /*** INIT PERFORMS PRINT ROUTINE INITIALIZATION ***/
108:     CALL INIT (PARM);
```

```
/* :LIST_&_SUM,REPORT=ROADLOG,DATA=XXXXX */

109:    /** SET UP COLUMN HEADINGS **/
110:    #_HDGS = 5;
111:    HEADING(3) = '              SECTION                          MILE    ' ||
112:       ' MTCE PROJECT      YEAR   SURF  THICKNESS   WIDTH   NO ' ||
113:       '  SECT   LOCA-  PROJCT   ACCUM';
114:    HEADING(4) = '            DESCRIPTION                       POST    ' ||
115:       ' SECT NUMBER     BLT IMP  TYPE  SURF BASE  SURF RDY LN ' ||
116:       ' LENGTH  TION   LENGTH  MILEAGE';

117:    /** READ TABLE OF COUNTY NAMES **/
118:    OPEN FILE (TABLE) INPUT RECORD TITLE ('CNTYTBL');
119:    DO J1=1 TO 56;
120:       READ FILE (TABLE) SET (PTR_TBL);
121:       COUNTY(J1) = RECORD;
122:       END;
123:    CLOSE FILE (TABLE);
124:    OPEN FILE (ROADLOG) INPUT SEQL;
125:    ON ENDFILE (ROADLOG) GO TO RETURN;
126:    READ FILE (ROADLOG) SET (PTR_RLG) KEY (STARTKEY);
127:    INIT_FLAG = 'I';
128:    J_CARRIAGE = 1;


129: /***** MAIN CONTROL LOOP *****/

130:    /** INIT_FLAG IS SET WHENEVER A ROUTE IS FINISHED, AND ITS COUNTY
131:        SUMMARY PRODUCED **/

132: LOOP:
133:    IF INIT_FLAG='I' THEN GOTO VARIABLE_INITIALIZER;
134: L05:
135:    IF RLG.REMARK='  ' | RLG.REMARK='ER' | RLG.REMARK='EN' |
136:       RLG.REMARK='DS' | RLG.REMARK='CO' | RLG.REMARK='OS' |
137:       RLG.REMARK='NE' | RLG.REMARK='SP' | RLG.REMARK='LP'
138:       THEN GOTO PRINT_RECORD;

139: READ_NEXT_RECORD:
140:    READ FILE (ROADLOG) SET (PTR_RLG);
141:    GOTO LOOP;


142: RETURN:
143:    CLOSE FILE (ROADLOG);
144:    CLOSE FILE (TABLE);
145:    CALL EXIT (PARM);
146:    RETURN;


147: /***** SUBROUTINE TO INITIALIZE VARIABLE AT THE BEGINNING OF ROUTE *

148: VARIABLE_INITIALIZER:
149:    INIT_FLAG = ' ';
150:    #_COUNTIES = 0;
151:    ACCUM_MILEAGE = 0;
152:    ACNTY = 0;
153:    DUPLICATE = 0;
154:    MBC = 0;
```

```
/* :LIST_&_SUM,REPORT=ROADLOG,DATA=XXXXX */

155:     OUT_OF_STATE = 0;
156:     TOT_CONST = 0;
157:     TOTALS = 0;
158:     ZRT_# = RLG.RT_#;
159:     IF RLG.SYSTEM='I' THEN SUBSTR(HEADING(1),46,45) =
160:        'FEDERAL AID INTERSTATE ROUTE NUMBER' || ZRT_#;
161:     ELSE IF RLG.SYSTEM='P' THEN SUBSTR(HEADING(1),46,45) =
162:        'FEDERAL AID PRIMARY ROUTE NUMBER' || ZRT_#;
163:     ELSE SUBSTR(HEADING(1),46,45) =
164:        'FEDERAL AID SECONDARY ROUTE NUMBER ' || ZRT_#;
165:     J1 = 5;
166:     IF PAGE_POSITION+20>PAGE_SIZE THEN PAGE_POSITION = PAGE_SIZE;
167:     ELSE DO J2=1 TO #_HDGS;
168:        PRINTER = HEADING(J2);
169:        CALL PRINTX (F(J1));
170:        J1 = 1;
171:        END;
172:     GOTO L05;


173: /***** SUBROUTINE TO FORMAT ROADLOG RECORDS AND PRINT THEM *****/

174: PRINT_RECORD:
175:     IF RLG.REMARK='DS' THEN DO;
176:        PRINTER = ' ';
177:        SUBSTR(PRINTER,60,35) = RLG.DESCR;
178:        #_LINES = 8;
179:        CALL PRINTXA (F(2),#_LINES);
180:        J_CARRIAGE = 2;
181:        GOTO READ_NEXT_RECORD;
182:        END;
183:     DFRLG = ' ';
184:     IF RLG.REMARK='ER' | RLG.REMARK='EN' | RLG.REMARK='CO'
185:        THEN DO;
186:        FRLG.DESCR = RLG.DESCR;
187:        IF RLG.REMARK¬='ER' THEN DO;
188:           FRLG.MILEPOST = RLG.MILEPOST;
189:           FRLG.OFFSET = RLG.OFFSET;
190:           END;
191:        PRINTER = DFRLG;
192:        CALL PRINTX (F(J_CARRIAGE));
193:        J_CARRIAGE = 1;
194:        IF RLG.REMARK='CO' THEN GOTO CO_PROCESSOR;
195:        IF RLG.REMARK='EN' THEN GOTO SUMMARY_BY_COUNTIES;
196:        GOTO READ_NEXT_RECORD;
197:        END;

198: NO_REMARK:
199:     FRLG = RLG, BY NAME;
200:     FRLG.MILEPOST = RLG.MILEPOST;
201:     FRLG.OFFSET = RLG.OFFSET;
202:     ACCUM_MILEAGE = ACCUM_MILEAGE + RLG.SECTN;
203:     FRLG.ACCUM_MLGE = ACCUM_MILEAGE;
204:     FRLG.SURTYP = SURF(RLG.SURF_TYPE_CODE);
205:     IF SUBSTR(FRLG.PROJ_CLASS,1,3)='MC'
206:        THEN FRLG.PROJ_CLASS = 'CITY CONSTR';
207:     ELSE IF SUBSTR(FRLG.PROJ_CLASS,1,3)='CC' THEN
208:        FRLG.PROJ_CLASS = 'CNTY CONSTR';
```

```
/* :LIST_&_SUM,REPORT=ROADLOG,DATA=XXXXX */

209:    IF RLG.POPULATION_CODE>3 & RLG.LOCN(1)=2
210:       THEN FRLG.LOCATION = 'URBAN';
211:     ELSE FRLG.LOCATION = LOC2(RLG.LOCN(1));
212:     PRINTER = DFRLG;
213:     CALL PRINTX (F(J_CARRIAGE));
214:     J_CARRIAGE = 1;

215:     /** CHECK FOR DUPLICATE MILEAGES **/
216:    IF RLG.LOCN(2)=0 THEN GOTO SAVE_MILEAGES;
217:     DFRLG = ' ';
218:     FRLG.LOCATION = LOC2(RLG.LOCN(2));
219:     PRINTER = DFRLG;
220:     CALL PRINTX (F(1));
221:     GOTO SAVE_MILEAGES;


222: /***** SUBROUTINE TO ACCUMULATE MILEAGES FOR COUNTY SUMMARY *****/

223: SAVE_MILEAGES:
224:     MLGE = 0;
225:     MLGE.ROUTE = RLG.ROUTE;
226:     MLGE.CONST = RLG.SECTN - RLG.WYE - RLG.UNIMP;
227:     MLGE.UNIMP = RLG.UNIMP;
228:     MLGE.WYE = RLG.WYE;
229:     DO J1=1 TO 2;
230:        IF RLG.LOCN(J1)=1 THEN MLGE.CITY=RLG.SECTN;
231:           ELSE IF RLG.LOCN(J1)=2 THEN MLGE.CNTY=RLG.SECTN;
232:           ELSE IF RLG.LOCN(J1)=3 THEN MLGE.NFOR=RLG.SECTN;
233:           ELSE IF RLG.LOCN(J1)=4 THEN MLGE.IRES=RLG.SECTN;
234:           ELSE IF RLG.LOCN(J1)¬=0 THEN MLGE.OTHR=RLG.SECTN;
235:        END;
236:     IF RLG.LOCN(2)¬=0 THEN DUPLICATE=DUPLICATE+RLG.SECTN;
237:     TOT_CONST = TOT_CONST + RLG.CONST;

238:     /** STORE THE MILEAGES BY COUNTY **/
239:     IF RLG.REMARK='OS' THEN DO;
240:        OUT_OF_STATE = OUT_OF_STATE + MLGE;
241:        GOTO READ_NEXT_RECORD;
242:        END;
243:     J1 = RLG.COUNTY_#;
244:     IF J1=0 | J1>56 THEN DO;
245:        CALL ASTER;
246:        PRINTER = '*** INVALID COUNTY NUMBER IN ABOVE RECORD';
247:        CALL PRINTX (F(2));
248:        CALL ASTER;
249:        GOTO READ_NEXT_RECORD;
250:        END;
251:     IF RLG.REMARK='SP' THEN J2 = 2;
252:        ELSE IF RLG.REMARK='LP' THEN J2 = 3;
253:        ELSE J2 = 1;
254:     IF ACNTY(J1,J2)=0 THEN DO;
255:        #_COUNTIES = #_COUNTIES + 1;
256:        IF #_COUNTIES>25 THEN DO;
257:           PRINTER = '*** INSUFFICIENT STORAGE FOR COUNTY SUMMARY';
258:           CALL PRINTX (F(2));
259:           GOTO RETURN;
260:           END;
261:        ACNTY(J1,J2) = #_COUNTIES;
```

```
/* :LIST_&_SUM,REPORT=ROADLOG,DATA=XXXXX */

262:        END;
263:     J1 = ACNTY(J1,J2);
264:     MBC(J1) = MBC(J1) + MLGE;
265:     GOTO READ_NEXT_RECORD;


266: /***** SUBROUTINE TO PROCESS COINCIDENT SECTIONS *****/

267: CO_PROCESSOR:
268:     SAVE_KEY = RLG2.KEY;
269:     KEY1 = RLG2.CN.RT_# || RLG2.CN.KEY(1);
270:     KEY2 = RLG2.CN.RT_# || RLG2.CN.KEY(2);
271:     ON KEY (ROADLOG) BEGIN;
272:        CALL ASTER;
273:        PRINTER = '*** NO RECORD FOR STARTKEY SPECIFIED';
274:        CALL PRINTX (F(2));
275:        CALL ASTER;
276:        GO TO BACK;
277:        END;
278:     READ FILE (ROADLOG) SET (PTR_RLG) KEY (KEY1);
279:     DO WHILE (RLG2.KEY<KEY2);
280:        ACCUM_MILEAGE = ACCUM_MILEAGE + RLG.SECTN;
281:        READ FILE (ROADLOG) SET (PTR_RLG);
282:        END;

283: BACK:
284:     DFRLG = ' ';
285:     FRLG.ACCUM_MLGE = ACCUM_MILEAGE;
286:     PRINTER = DFRLG;
287:     CALL PRINTX (F(0));
288:     J_CARRIAGE = 1;
289:     READ FILE (ROADLOG) SET (PTR_RLG) KEY (SAVE_KEY);
290:     GOTO READ_NEXT_RECORD;


291: /***** SUBROUTINE TO PRINT THE SUMMARY BY COUNTIES *****/

292: SUMMARY_BY_COUNTIES:
293:     #_HDGS = 2;
294:     PRINTER = ' ';
295:     SUBSTR(PRINTER,46,45)='********** SUMMARY BY COUNTIES **********';
296:     #_LINES = #_COUNTIES + 5;
297:     IF #_LINES>6 THEN #_LINES = #_LINES + 2;
298:     IF OUT_OF_STATE.ROUTE¬=0 THEN #_LINES = #_LINES + 2;
299:     CALL PRINTXA (F(2),#_LINES);
300:     PRINTER = '                              ROUTE  CONST   UNIMP   WYE ' ||
301:        '    CITY COUNTY  NATFOR  INDRES  OTHER   NET RURAL';
302:     CALL PRINTX (F(2));
303:     J_CARRIAGE = 2;
304:     J3 = 0;
305:     TOTALS = 0;
306:     DO J1=1 TO 3;
307:        DO J2=1 TO 56;
308:           IF ACNTY(J2,J1)=0 THEN GO TO NEXT_COUNTY;
309:           TOTALS = TOTALS + MBC(ACNTY(J2,J1));
310:           J3 = J3 + 1;
311:           ML = MBC(ACNTY(J2,J1));
312:           PRINTER = COUNTY(J2) || CODE(J1) || STRING_ML;
```

```
/* :LIST_&_SUM,REPORT=ROADLOG,DATA=XXXXX */

313:           CALL PRINTX (F(J_CARRIAGE));
314:           J_CARRIAGE = 1;
315: NEXT_COUNTY:
316:           END;
317:        END;
318:     IF J3>1 THEN DO;
319:        ML = TOTALS;
320:        PRINTER = '   TOTAL                ' || STRING_ML;
321:        CALL PRINTX (F(2));
322:        END;
323:     IF OUT_OF_STATE.ROUTE¬=0 THEN DO;
324:        ML = OUT_OF_STATE;
325:        PRINTER = '  + OUT OF STATE        ' || STRING_ML;
326:        CALL PRINTX (F(1));
327:        TOTALS = TOTALS + OUT_OF_STATE;
328:        ML = TOTALS;
329:        PRINTER = '   TOTAL                ' || STRING_ML;
330:        CALL PRINTX (F(1));
331:        END;
332:     RURAL_MILEAGE = TOTALS.CNTY + TOTALS.NFOR + TOTALS.IRES +
333:        TOTALS.OTHR - DUPLICATE - OUT_OF_STATE.CONST;
334:     PRINTER = ' ';
335:     SUBSTR(PRINTER,94,10) = RURAL_MILEAGE;
336:     CALL PRINTX (F(0));
337:     IF TOT_CONST¬=TOTALS.CONST THEN DO;
338:        CALL ASTER;
339:        PRINTER = '*** CONSTRUCTED MILEAGE IN ERROR:  ' || TOT_CONST;
340:        CALL PRINTX (F(2));
341:        CALL ASTER;
342:        END;
343:     IF PAGE_SIZE-PAGE_POSITION>2 THEN DO;
344:        PRINTER = LINES;
345:        CALL PRINTX (F(2));
346:        END;
347:     INIT_FLAG = 'I';
348:     #_HDGS = 5;
349:     IF SUBSTR(RLG2.KEY,1,4)>=ENDKEY THEN GOTO RETURN;
350:     GOTO READ_NEXT_RECORD;

351: END LISTSUM;
```

SURF-TYPE -- SURF-TYPE is implemented in two separate programs: NDR and NDR1. NDR1 is invoked when SUMMARY=RTE-NO is specified, unless DATA=ILOOP is also specified. NDR is invoked in all other cases.

MAIN PHASE:

Member Name	NDR	
Language	PL/I	
Subroutines	PRINTX1	
	SDC	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- SURF-TYPE output
	ROADLOG	-- Roadlog file
	PROJTBL	-- Project Class table
	SURFTBL	-- Surface Type table
	CITYTBL	-- City name table
	CNTYTBL	-- County name table
Instruction	1 - 3	"NDR"
	4	"1"/"2"/"3"/"4"/"5"/"6"/"7"/ for SUMMARY=RTE-NO/PROJ-#/COUNTY/ CITIES/YR-BLT/SUR-WD/YR-IMP
	5	"A"/"U" for MILEAGE=ALL/URBAN
	6	"I"/"P"/"S"/"C"/"A"/"L"/" " for DATA=INT/PRIM/SEC/INT+PRIM/ ALL/ILOOP/all-other
	40 - 52	Beginning key
	56 - 68	Ending key

NDR calculates and prints all of the surface type summaries other than SUMMARY=RTE-NO (it also produces SUMMARY=RTE-NO if DATA=ILOOP is specified). Out-of-state mileage ("OS" records) is not included in the summaries. The program operates in several modes, depending upon the options specified. These modes may be referred to as loop, urban, municipal, and normal modes. The program operates in loop mode whenever DATA=ILOOP is specified, requesting a summary of Interstate loop mileage. In this mode, the Primary system is scanned for "IL" (Interstate loop definition) records. When one is found, the beginning and ending keys of the loop are formed from the description, and the loop read. Each mileage record within the loop (NOTE: there is no out-of-state mileage on the Interstate system) is processed if MILEAGE=

ALL; only those occurring in a city of population 5000 or greater are processed if MILEAGE=URBAN. If DATA=ILOOP is not specified and MILEAGE=URBAN is specified, the program operates in urban mode. In this mode, the program begins searching at the point specified in the DATA parameter, and scans until the end point specified. Each record is tested for a population code of 4 or greater (5000 or more). Any such record is processed. Note that this method automatically eliminates descriptor records, which contain a population code of 0. No test is needed for out-of-state mileage, as there is no urban out-of-state mileage. Municipal mode is entered if neither DATA=ILOOP nor MILEAGE= URBAN is specified, but SUMMARY=CITIES is. In this mode, the program searches the data range for (mileage) records containing a location code of CITY (stored in the file as a decimal value of 1). Again, descriptor records are eliminated because these do not have location codes present. No test is needed for out-of-state mileage, as there is no municipal out-of-state mileage. Normal mode is used in all other cases. In this mode, the data range is searched for records containing " ," "SP," "LP," or "NE" codes (all mileage records other than "OS" out-of-state records). No matter which mode of operation is utilized, the procedure SAVE_MILEAGES is invoked each time a record is found for processing. This internal subroutine checks the 4th character of the instruction for the summary type, and sets a subscript (J1) to the value of the appropriate parameter. If SUMMARY=CITIES and either DATA=ILOOP or MILEAGE=URBAN is specified, SAVE_MILEAGES may be invoked with non-municipal records due to the mode of operation; hence, if J1 is zero in this case, an immediate return is taken. At all other times when J1 takes on a zero value, an error message is printed -- the parameter has not been coded on a mileage record. For each value the parameter may take on, 16 array elements may be required -- 8 for rural and 8 for municipal mileage. (Eight surface type categories are summarized on.) In the worst case (SUMMARY=CITIES), 126x8x2 decimal (7,3)

variables, or 8064 bytes, are required. However, when SUMMARY=CITIES is specified, only municipal mileage will be present, causing a direct-access method to set up twice as much core as will be needed. Hence, to save core, an indirect method is utilized. This method of storage allocation is explained above, under "LIST-&-SUM." In NDR, the array MILES is equivalent to LIST-&-SUM's array MBC, POINTERS is equivalent to ACNTY, and #_PARMS is equivalent to #_COUNTIES. After calculating J1 (based on the summarizing parameter), J2 is set to 1 for rural and 2 for municipal. Storage is then allocated if POINTERS (J1,J2) is zero. If non-zero, POINTERS(J1,J2) contains a pointer into MILES into which the mileage is to be added. After allocating storage, the program examines the Roadlog 4-digit surface type code (the larger code is used to allow other breakdowns than the usual one, such as for State Mileage reports), and sets J4 to a value between 1 and 8 as specified in the surface type table. The Roadlog section length is added into MILES, and a return is taken to the proper mode of operation. After scanning all of the Roadlog records in the data range, the summary is printed. Depending upon the SUMMARY parameter, the program may have to read one of the tables PROJTBL, CNTYTBL, or CITYTBL to obtain row headings. POINTERS is then scanned on the first subscript, ensuring that the parameter values will be printed from the smallest to the largest value. Each time a non-zero value is found, the row heading and the rural and municipal mileage (and a total) is set up in structure MLGE. This structure is passed to an external subroutine SDC (also used by NDR1), which performs the actual printing. As each line is set up for printing, the array TOTALS is accumulated, giving the total rural and municipal values in the summary. A final call to SDC prints the totals.

The SDC and NDR program listings follow:

```
SDC:  PROCEDURE (MILEAGES);

 1: SDC:  PROCEDURE (MILEAGES);


 2: /***** ROUTINE TO PRINT ONE SECTION OF SUMMARY                    ****
 3: /*****                                                           ****
 4: /***** THREE LINES ARE PRINTED:  ONE FOR RURAL, ONE FOR MUNIC,   *****
 5: /*****          AND ONE FOR TOTAL MILEAGE FOR THE GIVEN PARAMETER.*****
 6: /***** MILEAGES AND ROW HEADING ARE PASSED BY CALLING PROCEDURE ****


 7: /***** DECLARATION OF VARIABLES (ALPHABETICAL ORDER) *****/

 8: DECLARE
 9:    #_LINES PIC'ZZ',
10:    F(0:9) PIC'Z' INIT(0,1,2,3,4,5,6,7,8,9),
11:    MILEAGES CHAR(160),
12:    1  MLGE DEF MILEAGES,
13:       3  HDG CHAR(20),
14:       3  M(9,2) PIC'ZZZZVZZZ',
15:    PRINTER CHAR(132) EXT,
16:    STRING_STRCT CHAR(115) DEF STRCT,
17:    1  STRCT,
18:       3  HDG CHAR(20),
19:       3  RUR_MUN CHAR(5),
20:       3  S(9) PIC'ZZZZZZV.ZZZ';


21: /***** ROUTINE INITIALIZATION *****/


22:    ON ERROR BEGIN;
23:       PRINTER = '*** TERMINAL ERROR IN SRTYPP PHASE';
24:       CALL PRINTX (F(2));
25:       GO TO RETURN;
26:       END;


27: /***** MAIN PROGRAM BODY *****/
28:    IF MLGE.M(9,1)¬=0 THEN J1 = 1;  ELSE J1 = 0;
29:    IF MLGE.M(9,2)¬=0 THEN J2 = 1;  ELSE J2 = 0;
30:    IF J1+J2>1 THEN #_LINES = 4;  ELSE #_LINES = 2;
31:    PRINTER = ' ';
32:    CALL PRINTXA (F(1),#_LINES);
33:    STRCT.HDG = MLGE.HDG;
34:    IF J1=1 THEN DO;
35:       STRCT.RUR_MUN = 'RURAL';
36:       STRCT.S = MLGE.M(*,1);
37:       PRINTER = STRING_STRCT;
38:       CALL PRINTX (F(1));
39:       STRCT.HDG = ' ';
40:       END;
41:    IF J2=1 THEN DO;
42:       STRCT.RUR_MUN = 'MUNIC';
43:       STRCT.S = MLGE.M(*,2);
44:       PRINTER = STRING_STRCT;
45:       CALL PRINTX (F(1));
46:       END;
47:    IF J1+J2>1 THEN DO;
```

```
SDC:  PROCEDURE (MILEAGES);

 48:        STRCT.RUR_MUN = 'TOTAL';
 49:        STRCT.S = MLGE.M(*,1) + MLGE.M(*,2);
 50:        PRINTER = STRING_STRCT;
 51:        CALL PRINTX (F(1));
 52:        END;

 53: RETURN:

 54: END SDC;
```

```
/* :SURF_TYPE,REPORT=ROADLOG,SUMMARY=XXXXX,DATA=XXXXX,MILEAGE=XXXXX */

 1: /* :SURF_TYPE,REPORT=ROADLOG,SUMMARY=XXXXX,DATA=XXXXX,MILEAGE=XXXXX
 2: SURFTYP:  PROCEDURE (PARM) OPTIONS (MAIN);

 3: /* PRINT SUBROUTINE */
 4: DECLARE
 5:    PARM CHAR(100),
 6:    INSTR CHAR(80) EXT,
 7:    #_HDGS PIC'Z' DEF INSTR POS(72),
 8:    BLANKS CHAR(132) STATIC INIT (' '),
 9:    F(0:9) PIC'Z' STATIC INIT (0,1,2,3,4,5,6,7,8,9),
10:    HEADING(9) CHAR(132) EXT,
11:    PRINTER CHAR(132) EXT;

12: /* INSTRUCTION */
13: DECLARE
14:    REPORT CHAR(1) DEF INSTR POS(3),
15:    SUMMARY CHAR(1) DEF INSTR POS(4),
16:    SUMMARY_# PIC'Z' DEF INSTR POS(4),
17:    URBAN_IND CHAR(1) DEF INSTR POS(5),
18:    SYSTEM CHAR(1) DEF INSTR POS(6),
19:    STARTKEY CHAR(16) DEF INSTR POS(40),
20:    ENDKEY CHAR(16) DEF INSTR POS(56);

21: /* ROADLOG FILE */
22: DECLARE
23:    1  RLG BASED (PTR_RLG),
24:       3  DUM1 CHAR(1),
25:       3  SYSTEM CHAR(1),
26:       3  RT_# PIC'999',
27:       3  MILEPOST PIC'ZZZ',
28:       3  PLUS CHAR(1),
29:       3  OFFSET PIC'ZV.ZZZ',
30:       3 REMARK CHAR(2),
31:       3 (SECTN,ROUTE,CONST,UNIMP) DEC FIXED(5,3),
32:       3 WYE DEC FIXED(3,3),
33:       3  DESCR CHAR(35),
34:       3 PROJECT_CLASS CHAR(11),
35:       3 DIVIDED_CODE CHAR(1),
36:       3 (#_LANES,POPULATION_CODE) DEC FIXED(1,0),
37:       3 (CITY_#,COUNTY_#,YR_BLT,YR_IMP,FORHWY_#,ADMIN_CODE,
38:         LOCN(2),PROJ_CLASS,SURF_WIDTH,RDWAY_WIDTH) DEC FIXED(3,0),
39:       3 (SURF_THICKNESS,BASE_THICKNESS) DEC FIXED(3,1),
40:       3 DUM4 CHAR(1),
41:       3 (SURF_TYPE,MAINT_SEC) DEC FIXED(5,0),
42:       3 DATE,
43:         5 (MONTH,DAY,YEAR) DEC FIXED(3,0),
44:       3 DUM3 CHAR(2),
45:    1  RLG2 BASED (PTR_RLG),
46:       3  DUM1 CHAR(1),
47:       3  KEY CHAR(13),
48:       3 DUM2 CHAR(16),
49:       3  CN,
50:          5  DUM1 CHAR(5),
51:          5  RT_# CHAR(4),
52:          5  DUM2 CHAR(6),
53:          5  KEY(2) CHAR(10),
54:    ROADLOG FILE RECORD KEYED ENV(INDEXED);
```

```
/* :SURF_TYPE,REPORT=ROADLOG,SUMMARY=XXXXX,DATA=XXXXX,MILEAGE=XXXXX */

 55: /* OTHER VARIABLES */
 56: DECLARE
 57:    MAX_#_PARMS DEC FIXED (3) STATIC INIT (130),
 58:    MILES(130,8) DEC FIXED (7,3) STATIC,
 59:    POINTERS(130,2) DEC FIXED (3) STATIC,
 60:   (#_PARMS,#_PROJS) DEC FIXED (3) STATIC INIT (0),
 61:    #_SURF_TYPES DEC FIXED (2) STATIC,
 62:    SURTYP(50,2) DEC FIXED (4) STATIC,
 63:    C4 CHAR(4) STATIC,
 64:    PRJ CHAR(5) BASED (PTR_TBL),
 65:    PROJ(70) CHAR(4) STATIC;


 66: /***** INITIALIZATION *****/

 67:    CALL INIT (PARM);
 68:    CALL SET_HDGS;

 69:   /* IF SUMMARY=PROJ_# READ PROJTBL */
 70:    IF SUMMARY='2' THEN DO;
 71:       OPEN FILE (TABLE) INPUT RECORD TITLE ('PROJTBL');
 72:       ON ENDFILE (TABLE) GOTO FINPROJ;
 73:       DO #_PROJS=1 TO 70;
 74:          READ FILE (TABLE) SET (PTR_TBL);
 75:          PROJ(#_PROJS) = PRJ;
 76:          END;
 77: FINPROJ:
 78:       CLOSE FILE (TABLE);
 79:       END;

 80:    /* INITIALIZE ROADLOG FILE */
 81:    OPEN FILE (ROADLOG) INPUT SEQL;
 82:    ON ENDFILE (ROADLOG) GOTO PRINT_SUMMARY;
 83:    READ FILE (ROADLOG) SET (PTR_RLG) KEY (STARTKEY);

 84:    /* INIT VAR */
 85:    MILES = 0;
 86:    #_PARMS = 0;
 87:    POINTERS = 0;

 88:    /* CHECK FOR SUMMARY TYPE */
 89:    IF SYSTEM='L' THEN GOTO INT_LOOPS;
 90:    IF URBAN_IND='U' THEN GOTO URBAN;
 91:    IF SUMMARY='4' THEN GOTO CITY;


 92: /***** EXECUTION LOOPS *****/

 93:    DO WHILE (RLG2.KEY<=ENDKEY);
 94:       IF RLG.REMARK='  ' | RLG.REMARK='LP' | RLG.REMARK='SP' |
 95:          RLG.REMARK='NE'
 96:          THEN CALL SAVE_MILEAGES;
 97:       READ FILE (ROADLOG) SET (PTR_RLG);
 98:       END;
 99:    GOTO PRINT_SUMMARY;

100: URBAN:
```

```
/* :SURF_TYPE,REPORT=ROADLOG,SUMMARY=XXXXX,DATA=XXXXX,MILEAGE=XXXXX */

101:     DO WHILE (RLG2.KEY<=ENDKEY);
102:        IF RLG.POPULATION_CODE>=4 THEN CALL SAVE_MILEAGES;
103:        READ FILE (ROADLOG) SET (PTR_RLG);
104:        END;
105:     GOTO PRINT_SUMMARY;

106: CITY:
107:     DO WHILE (RLG2.KEY<=ENDKEY);
108:           IF RLG.LOCN(1)=1 THEN CALL SAVE_MILEAGES;
109:        READ FILE (ROADLOG) SET (PTR_RLG);
110:        END;
111:     GOTO PRINT_SUMMARY;

112: INT_LOOPS:
113:     DO WHILE (RLG2.KEY<=ENDKEY);
114:        IF RLG.REMARK='IL' THEN CALL IL_RECORD;
115:        READ FILE (ROADLOG) SET (PTR_RLG);
116:        END;

117: PRINT_SUMMARY:
118:     CLOSE FILE (ROADLOG);
119:     CALL GEN_SUMMARY;

120: RETURN:
121:     CLOSE FILE (ROADLOG);
122:     CALL EXIT (PARM);
123:     RETURN;


124: /***** SUBROUTINE TO CALCULATE HEADINGS *****/

125: SET_HDGS:  PROCEDURE;

126: DECLARE
127:     B1(7) DEC FIXED (2) STATIC INIT (41,41,44,45,42,41,41),
128:     B2(12) DEC FIXED (2) STATIC INIT
129:       (42,40,40,44,44,44,  42,38,41,44,45,41),
130:    (J1,J2) DEC FIXED,
131:     1  STYP BASED (PTR_TBL),
132:        3  S(2) PIC'ZZZZ',
133:        3  DUM CHAR(7),
134:        3  HD(2) CHAR(12),
135:     SUM_HDG(7) CHAR(13) STATIC INIT
136:        ('ROUTE NUMBER','PROJECT CLASS','COUNTY','CITY',
137:        'YEAR BUILT','SURFACE WIDTH','YEAR IMPROVED'),
138:     SUM1(7) CHAR(20) STATIC INIT (
139:        '                ROUTE','          PROJECT',' ',' ',
140:        '            YEAR','          SURFACE','          YEAR'),
141:     SUM2(7) CHAR(20) STATIC INIT (
142:        '              NUMBER','           CLASS',
143:        '   COUNTY','   CITY','          BUILT',
144:        '            WIDTH','         IMPROVED'),
145:     SYS_HDG(12) CHAR(50) STATIC INIT (
146:        '     FEDERAL AID INTERSTATE SYSTEM',
147:        'FEDERAL AID PRIMARY SYSTEM--LESS INTERSTATE',
148:        'FEDERAL AID PRIMARY AND INTERSTATE SYSTEMS',
149:        '    FEDERAL AID SECONDARY SYSTEM',
150:        '             ALL SYSTEMS',
```

```
/* :SURF_TYPE,REPORT=ROADLOG,SUMMARY=XXXXX,DATA=XXXXX,MILEAGE=XXXXX */

151:        '   NON-CHARGEABLE PARALLEL MILEAGE',
152:        '     FEDERAL AID INTERSTATE URBAN',
153:        'FEDERAL AID PRIMARY URBAN--LESS INTERSTATE URBAN',
154:        'FEDERAL AID PRIMARY AND INTERSTATE URBAN',
155:        '    FEDERAL AID SECONDARY URBAN',
156:        '              ALL URBAN',
157:        '       INTERSTATE LOOPS--URBAN');
158:     SUBSTR(HEADING(1),51,24) = 'SUMMARY OF SURFACE TYPES';
159:     HEADING(2) = SUBSTR(BLANKS,1,B1(SUMMARY_#)) ||
160:        'NET CONSTRUCTED LENGTH -- BY ' || SUM_HDG(SUMMARY_#);
161:     IF SYSTEM='I' THEN J1 = 1;
162:        ELSE IF SYSTEM='P' THEN J1 = 2;
163:        ELSE IF SYSTEM='C' THEN J1 = 3;
164:        ELSE IF SYSTEM='S' THEN J1 = 4;
165:        ELSE IF SYSTEM='A' THEN J1 = 5;
166:        ELSE IF SYSTEM='L' THEN J1 = 6;
167:        ELSE J1 = 0;
168:     IF J1¬=0 & URBAN_IND='U' THEN J1 = J1 + 6;
169:     IF J1¬=0 THEN HEADING(3) = SUBSTR(BLANKS,1,B2(J1)) || SYS_HDG(J1);
170:     HEADING(5) = SUM1(SUMMARY_#);
171:     SUBSTR(HEADING(5),109,5) = 'TOTAL';
172:     HEADING(6) = SUM2(SUMMARY_#);
173:     #_HDGS = 7;

174:     /* READ SURFACE TYPE TABLE */
175:     IF REPORT¬='Q'
176:        THEN OPEN FILE (TABLE) INPUT RECORD TITLE ('SURFTBL');
177:        ELSE OPEN FILE (TABLE) INPUT RECORD TITLE ('SMSFTBL');
178:     ON ENDFILE (TABLE) GOTO CONTINUE;
179:     SURTYP = 0;
180:     DO #_SURF_TYPES=1 TO 50;
181:        READ FILE (TABLE) SET (PTR_TBL);
182:        SURTYP(#_SURF_TYPES,*) = STYP.S;
183:        J2 = 18 + STYP.S(2)*10;
184:        SUBSTR(HEADING(5),J2,10)=STYP.HD(1);
185:        SUBSTR(HEADING(6),J2,10)=STYP.HD(2);
186:        END;
187: CONTINUE:
188:     #_SURF_TYPES = #_SURF_TYPES - 1;
189:     CLOSE FILE (TABLE);
190:     END SET_HDGS;


191: /***** SUBROUTINE TO ACCUMULATE MILEAGES *****/

192: SAVE_MILEAGES:  PROCEDURE;

193: DECLARE (J1,J2,J3,J4) DEC FIXED;

194:     IF SUMMARY='1' THEN J1 = RLG.RT_#;
195:           ELSE IF SUMMARY='2' THEN J1 = RLG.PROJ_CLASS;
196:        ELSE IF SUMMARY='3' THEN J1 = RLG.COUNTY_#;
197:        ELSE IF SUMMARY='4' THEN J1 = RLG.CITY_#;
198:        ELSE IF SUMMARY='5' THEN J1 = RLG.YR_BLT;
199:        ELSE IF SUMMARY='6' THEN J1 = RLG.SURF_WIDTH;
200:        ELSE IF SUMMARY='7' THEN J1 = RLG.YR_IMP;
201:     IF J1=0 THEN DO;
202:        IF SUMMARY='4' THEN RETURN;
```

```
/* :SURF_TYPE,REPORT=ROADLOG,SUMMARY=XXXXX,DATA=XXXXX,MILEAGE=XXXXX */

203:        PRINTER = '*** PARAMETER HAS ZERO VALUE AT KEY = ' || RLG2.KEY
204:        CALL PRINTX (F(1));
205:        J1 = 1;
206:        END;
207:      IF RLG.LOCN(1)=1
208:        THEN J2 = 2;
209:        ELSE J2 = 1;

210:     /* ALLOCATE STORAGE IF NECESSARY */
211:     IF POINTERS(J1,J2)=0 THEN DO;
212:        #_PARMS = #_PARMS + 1;
213:        IF #_PARMS>MAX_#_PARMS THEN DO;
214:           PRINTER = '*** STORAGE ALLOCATION EXCEEDED';
215:           CALL PRINTX (F(3));
216:           GOTO RETURN;
217:           END;
218:        POINTERS(J1,J2) = #_PARMS;
219:        END;

220:     /* GET SURFACE TYPE */
221:     J4 = RLG.SURF_TYPE;
222:     DO J3=1 TO #_SURF_TYPES;
223:        IF J4=SURTYP(J3,1) THEN GOTO SATISFIED_SURFACE_TYPE;
224:        END;
225:     PRINTER = '*** SURFACE TYPE (' || RLG.SURF_TYPE ||
226:        ') UNKNOWN AT KEY = ' || RLG2.KEY;
227:     CALL PRINTX (F(1));
228:     J3 = 1;
229:     GOTO ADD;
230: SATISFIED_SURFACE_TYPE:
231:     J3 = SURTYP(J3,2);

232: ADD:
233:     J1 = POINTERS(J1,J2);
234:     MILES(J1,J3) = MILES(J1,J3) + RLG.SECTN;
235:     END SAVE_MILEAGES;


236: /***** SUBROUTINE TO PROCESS INTERSTATE LOOPS *****/

237: IL_RECORD:  PROCEDURE;

238: DECLARE (KEY1,KEY2,SAVE_KEY) CHAR(16);
239:     SAVE_KEY = RLG2.KEY;
240:     KEY1 = RLG2.CN.RT_# || RLG2.CN.KEY(1);
241:     KEY2 = RLG2.CN.RT_# || RLG2.CN.KEY(2);
242:     ON KEY (ROADLOG) BEGIN;
243:        PRINTER = '*** NO RECORD FOR ILOOP SECTN ' || KEY1 ||
244:           ' AT ROADLOG RECORD ' || SAVE_KEY;
245:        CALL PRINTX (F(1));
246:        GOTO BACK;
247:        END;
248:     READ FILE (ROADLOG) SET (PTR_RLG) KEY (KEY1);
249:     DO WHILE (RLG2.KEY<KEY2);
250:        IF RLG.SECTN¬=0 & (URBAN_IND¬='U' | RLG.POPULATION_CODE>=4)
251:           THEN CALL SAVE_MILEAGES;
252:        READ FILE (ROADLOG) SET (PTR_RLG);
253:        END;
```

```
/* :SURF_TYPE,REPORT=ROADLOG,SUMMARY=XXXXX,DATA=XXXXX,MILEAGE=XXXXX */


254: BACK:
255:    READ FILE (ROADLOG) SET (PTR_RLG) KEY (SAVE_KEY);
256:    END IL_RECORD;


257: /***** SUBROUTINE TO FORMAT & PRINT SUMMARY *****/

258: GEN_SUMMARY:  PROCEDURE;

259: DECLARE
260:    CHD(132) CHAR(20),
261:    DDN CHAR(8),
262:   (J1,J2,J3,J4) DEC FIXED,
263:    MILEAGES CHAR(160),
264:    1  MLGE DEF MILEAGES,
265:       3  HDG CHAR(20),
266:       3  M(9,2) PIC'ZZZZVZZZ',
267:    NHD(132) PIC'BBBBBBBBBBZZZZZBBBBB',
268:    TOTALS(9,2) DEC FIXED (7,3),
269:    RECORD CHAR(80) BASED (PTR_TBL);

270:    CHD = ' ';
271:    IF SUMMARY='1' | SUMMARY>'4' THEN DO J1=1 TO 100;
272:       IF SUMMARY='5' | SUMMARY='7'
273:          THEN NHD(J1) = J1 + 1900;
274:          ELSE NHD(J1) = J1;
275:       CHD(J1) = NHD(J1);
276:       END;
277:    ELSE IF SUMMARY='2' THEN DO J1=1 TO #_PROJS;
278:       SUBSTR(CHD(J1),12,4) = PROJ(J1);
279:       END;
280:    ELSE DO;
281:       IF SUMMARY='3'
282:          THEN DO;  J2 = 15;  DDN = 'CNTYTBL';  END;
283:          ELSE DO;  J2 = 18;  DDN = 'CITYTBL';  END;
284:       OPEN FILE (TABLE) INPUT RECORD TITLE (DDN);
285:       ON ENDFILE (TABLE) GOTO CLOSE_TABLE;
286:       DO J3=1 TO 200;
287:          READ FILE (TABLE) SET (PTR_TBL);
288:          CHD(J3) = SUBSTR(RECORD,1,J2);
289:          END;
290: CLOSE_TABLE:
291:          CLOSE FILE (TABLE);
292:       END;

293:    /* PRINT THE SUMMARY */
294:    TOTALS = 0;
295:    DO J1=1 TO MAX_#_PARMS;
296:       IF POINTERS(J1,1)¬=0 | POINTERS(J1,2)¬=0 THEN DO;
297:          MLGE.M = 0;
298:          DO J2=1 TO 2;
299:             IF POINTERS(J1,J2)=0 THEN GOTO IGNORE;
300:             J3 = POINTERS(J1,J2);
301:             DO J4=1 TO 8;
302:                MLGE.M(J4,J2) = MILES(J3,J4);
303:                MLGE.M(9,J2) = MLGE.M(9,J2) + MILES(J3,J4);
304:                END;
```

```
/* :SURF_TYPE,REPORT=ROADLOG,SUMMARY=XXXXX,DATA=XXXXX,MILEAGE=XXXXX */

305: IGNORE:     END;
306:           TOTALS = TOTALS + MLGE.M;
307:           MLGE.HDG = CHD(J1);
308:           CALL SDC (MILEAGES);
309:           END;
310:       END;

311: /* PRINT TOTALS */
312:    MLGE.HDG = '              TOTAL';
313:    PRINTER = ' ';
314:    CALL PRINTX (F(1));
315:    MLGE.M = TOTALS;
316:    CALL SDC (MILEAGES);
317:    END GEN_SUMMARY;

318: END SURFTYP;
```

SUMMARY BY ROUTE NUMBER PHASE:

Member Name	NDR1	
Language	PL/I	
Subroutines	PRINTX1	
	SDC	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- SURF-TYPE output
	ROADLOG	-- Roadlog file
	SURFTBL	-- Surface type table
Instruction	1 - 4	"NDR1"
	5	"A"/"U" for MILEAGE=ALL/URBAN
	6	"I"/"P"/"S"/"C"/"A"/" " for DATA=INT/PRIM/SEC/INT+PRIM/ ALL/all-other
	40 - 52	Beginning Roadlog key
	56 - 68	Ending Roadlog key

A separate phase has been established for summary by routes in order to conserve core storage. Because of the large number of Secondary routes, a large amount of storage would be required for a summary by routes when DATA=SEC or DATA=ALL is specified. In the worst case (DATA=ALL), the amount of storage would be about 400x2x8 decimal (7,3) variables, or 25,600 bytes of core. However, because the route system and number comprises the first portion of the Roadlog key, it is not necessary to store all of the values in core simultaneously. One complete route may be scanned, and its values printed. This method will not operate when DATA=ILOOP is specified, as the "IL" records do not appear in order by Interstate route number, but by primary route number. However, the ILOOP summary by routes can be easily obtained by NDR. NDR1 operates essentially like NDR. It may operate in either urban or normal mode (loop and municipal mode are not applicable). However, the program is much simpler due to the fact that no indirect storage acquisition is required. NDR1 keys on "EN" records to print a line of output. Subroutine SDC is utilized for performing the actual printing -- a structure with the route number and the rural and municipal mileage for that route is set up and passed.

The NDR1 program listing follows:

```
/* :SURF_TYPE,REPORT=ROADLOG,SUMMARY=RTE_NO,DATA=XXXXX,MILEAGE=XXXXX */

 1: /* :SURF_TYPE,REPORT=ROADLOG,SUMMARY=RTE_NO,DATA=XXXXX,MILEAGE=XXXXX
 2: SURFTYP:  PROCEDURE (PARM) OPTIONS (MAIN);

 3: /* PRINT SUBROUTINE */
 4: DECLARE
 5:    PARM CHAR(100),
 6:    INSTR CHAR(80) EXT,
 7:    #_HDGS PIC'Z' DEF INSTR POS(72),
 8:    BLANKS CHAR(132) STATIC INIT (' '),
 9:    F(0:9) PIC'Z' STATIC INIT (0,1,2,3,4,5,6,7,8,9),
10:    HEADING(9) CHAR(132) EXT,
11:    PRINTER CHAR(132) EXT;

12: /* INSTRUCTION */
13: DECLARE
14:    REPORT CHAR(1) DEF INSTR POS(3),
15:    URBAN_IND CHAR(1) DEF INSTR POS(5),
16:    SYSTEM CHAR(1) DEF INSTR POS(6),
17:    STARTKEY CHAR(16) DEF INSTR POS(40),
18:    ENDKEY CHAR(16) DEF INSTR POS(56);

19: /* ROADLOG FILE */
20: DECLARE
21:    1  RLG BASED (PTR_RLG),
22:       3  DUM1 CHAR(1),
23:       3  SYSTEM CHAR(1),
24:       3  RT_# PIC'999',
25:       3  MILEPOST PIC'ZZZ',
26:       3  OFFSET CHAR(6),
27:       3 REMARK CHAR(2),
28:       3 (SECTN,ROUTE,CONST,UNIMP)DEC FIXED(5,3),
29:       3  WYE DEC FIXED(3,3),
30:       3 DESCR CHAR(35),
31:       3 PROJ_CLASS CHAR(11),
32:       3 DIVIDED_CODE CHAR(1),
33:       3 (#_LANES,POPULATION_CODE)DEC FIXED(1,0),
34:       3 (CITY_#,COUNTY_#,YR_BLT,YR_IMP,FORHWY_#,ADMIN_CODE,
35:         LOCN(2),PROJ_CL,SURF_WIDTH,RDWAY_WIDTH) DEC FIXED(3,0),
36:       3 (SURF_THICKNESS,BASE_THICKNESS) DEC FIXED(3,1),
37:       3 SURF_TYPE_CODE DEC FIXED(1,0),
38:       3 (SURF_TYPE,MAINT_SEC) DEC FIXED(5,0),
39:       3 DATE,
40:          5 (MONTH,DAY,YEAR) DEC FIXED(3,0),
41:    1  RLG2 BASED (PTR_RLG),
42:       3  DUM1 CHAR(1),
43:       3  KEY CHAR(13),
44:    ROADLOG FILE RECORD KEYED ENV(INDEXED);

45: /* OTHER VARIABLES */
46: DECLARE
47:    SURTYP(50,2) DEC FIXED (4) STATIC,
48:    #_SURF_TYPES DEC FIXED (2) STATIC,
49:    MILEAGES CHAR(160) STATIC,
50:    1  MLGE DEF MILEAGES,
51:       3  HDG CHAR(20),
52:       3  M(9,2) PIC'ZZZZVZZZ',
53:    TOTALS(9,2) DEC FIXED (7,3) STATIC,
54:    TOTALS1(9,2) DEC FIXED (7,3) STATIC,
```

```
/* :SURF_TYPE,REPORT=ROADLOG,SUMMARY=RTE_NO,DATA=XXXXX,MILEAGE=XXXXX */

 55:    SYS_HDG(5) CHAR(43) STATIC INIT (
 56:       '     FEDERAL AID INTERSTATE SYSTEM',
 57:       'FEDERAL AID PRIMARY SYSTEM--LESS INTERSTATE',
 58:       'FEDERAL AID PRIMARY AND INTERSTATE SYSTEMS',
 59:       '    FEDERAL AID SECONDARY SYSTEM',
 60:       '              ALL SYSTEMS'),
 61:    B(5) DEC FIXED (2) STATIC INIT (42,40,40,44,44),
 62:    1  STYP BASED (PTR_TBL),
 63:       3  S(2) PIC'ZZZZ',
 64:       3  DUM CHAR(7),
 65:       3  HD(2) CHAR(12),
 66:   (J1,J2,J3) DEC FIXED STATIC;


 67: /***** INITIALIZATION *****/
 68:    CALL INIT (PARM);

 69:    /* SET UP HEADINGS */
 70:    SUBSTR(HEADING(1),51,24) = 'SUMMARY OF SURFACE TYPES';
 71:    SUBSTR(HEADING(2),42,41) =
 72:       'NET CONSTRUCTED LENGTH -- BY ROUTE NUMBER';
 73:    IF SYSTEM='I' THEN J1 = 1;
 74:       ELSE IF SYSTEM='P' THEN J1 = 2;
 75:       ELSE IF SYSTEM='C' THEN J1 = 3;
 76:       ELSE IF SYSTEM='S' THEN J1 = 4;
 77:       ELSE IF SYSTEM='A' THEN J1 = 5;
 78:       ELSE J1 = 0;
 79:    IF J1¬=0 THEN HEADING(3) = SUBSTR(BLANKS,1,B(J1)) || SYS_HDG(J1);
 80:    IF URBAN_IND='U' THEN DO;
 81:       SUBSTR(HEADING(4),55,12) = 'URBAN SYSTEM';
 82:       J1 = 6;
 83:       END;
 84:    ELSE J1 = 5;
 85:    SUBSTR(HEADING(J1),16,5) = 'ROUTE';
 86:    SUBSTR(HEADING(J1),109,5) = 'TOTAL';
 87:    SUBSTR(HEADING(J1+1),16,6) = 'NUMBER';
 88:    #_HDGS = J1 + 2;

 89:    /* READ TABLE OF SURFACE TYPES */
 90:    IF REPORT¬='Q'
 91:       THEN OPEN FILE (TABLE) INPUT RECORD TITLE ('SURFTBL');
 92:       ELSE OPEN FILE (TABLE) INPUT RECORD TITLE ('SMSFTBL');
 93:    ON ENDFILE (TABLE) GOTO CONTINUE;
 94:    SURTYP = 0;
 95:    DO #_SURF_TYPES=1 TO 50;
 96:       READ FILE (TABLE) SET (PTR_TBL);
 97:       SURTYP(#_SURF_TYPES,*) = STYP.S;
 98:       J2 = 18 + STYP.S(2)*10;
 99:       SUBSTR(HEADING(J1),J2,10) = STYP.HD(1);
100:       SUBSTR(HEADING(J1+1),J2,10) = STYP.HD(2);
101:       END;
102: CONTINUE:
103:    #_SURF_TYPES = #_SURF_TYPES - 1;
104:    CLOSE FILE (TABLE);

105:    /* INITIALIZE ROADLOG FILE */
106:    ON ENDFILE (ROADLOG) GOTO FINISH_SUMMARY;
107:    READ FILE (ROADLOG) SET (PTR_RLG) KEY (STARTKEY);
```

```
/* :SURF_TYPE,REPORT=ROADLOG,SUMMARY=RTE_NO,DATA=XXXXX,MILEAGE=XXXXX */

108:     /* INIT VAR */
109:     MILEAGE_IND = 0;
110:     MILEAGES = ' ';
111:     TOTALS = 0;

112:     /* CHECK FOR URBAN SUMMARY */
113:     IF URBAN_IND='U' THEN GOTO URBAN;


114: /***** EXECUTION LOOPS *****/

115:     DO WHILE (RLG2.KEY<=ENDKEY);
116:        IF RLG.REMARK='  ' | RLG.REMARK='LP' | RLG.REMARK='SP' |
117:           RLG.REMARK='NE'
118:           THEN CALL SAVE_MILEAGES;
119:        ELSE IF RLG.REMARK='EN' & MILEAGE_IND=1 THEN CALL PRINT_LINE;
120:        READ FILE (ROADLOG) SET (PTR_RLG);
121:        END;
122:     GOTO FINISH_SUMMARY;

123: URBAN:
124:     DO WHILE (RLG2.KEY<=ENDKEY);
125:        IF RLG.POPULATION_CODE>=4 THEN CALL SAVE_MILEAGES;
126:        ELSE IF RLG.REMARK='EN' & MILEAGE_IND=1 THEN CALL PRINT_LINE;
127:        READ FILE (ROADLOG) SET (PTR_RLG);
128:        END;

129: FINISH_SUMMARY:
130:     IF MILEAGE_IND=1 THEN CALL PRINT_LINE;
131:     IF SYSTEM='C' THEN DO;
132:        MLGE.HDG = '  SUBTOTAL PRIMARY';
133:        MLGE.M = TOTALS;
134:        CALL SDC (MILEAGES);
135:        TOTALS = TOTALS + TOTALS1;
136:        END;
137:     MLGE.HDG = '                TOTALS';
138:     MLGE.M = TOTALS;
139:     CALL SDC (MILEAGES);

140: RETURN:
141:     CLOSE FILE (ROADLOG);
142:     CALL EXIT (PARM);
143:     RETURN;


144: /***** SUBROUTINE TO ACCUMULATE MILEAGES *****/

145: SAVE_MILEAGES:  PROCEDURE;
146:      DECLARE  (J1,J2,J3) DEC FIXED;
147:      IF RLG.LOCN(1)=1
148:        THEN J3 = 2;
149:        ELSE J3 = 1;
150:     J1 = RLG.SURF_TYPE;
151:     DO J2=1 TO #_SURF_TYPES;
152:        IF J1=SURTYP(J2,1) THEN GOTO SATISFIED_SURFACE_TYPE;
153:        END;
154:     PRINTER = '*** SURFACE TYPE (' || RLG.SURF_TYPE ||
```

```
/* :SURF_TYPE,REPORT=ROADLOG,SUMMARY=RTE_NO,DATA=XXXXX,MILEAGE=XXXXX */

155:        ') UNKNOWN AT KEY = ' || RLG2.KEY;
156:     CALL PRINTX (F(1));
157:     J2 = 1;
158:     GOTO ADD;
159: SATISFIED_SURFACE_TYPE:
160:     J2 = SURTYP(J2,2);

161: ADD:
162:     MLGE.M(J2,J3) = MLGE.M(J2,J3) + RLG.SECTN;
163:     MILEAGE_IND = 1;
164:     END SAVE_MILEAGES;


165: /***** SUBROUTINE TO PRINT ONE ROUTE *****/

166: PRINT_LINE:  PROCEDURE;
167:      DECLARE J1 DEC FIXED;
168:    MLGE.HDG=SUBSTR(BLANKS,1,16)||SUBSTR(RLG2.KEY,2,3);
169:     IF SUBSTR(MLGE.HDG,17,2)='00' THEN SUBSTR(MLGE.HDG,17,2) = ' ';
170:     IF SUBSTR(MLGE.HDG,17,1)='0' THEN SUBSTR(MLGE.HDG,17,1) = ' ';
171:     DO J1=1 TO 8;
172:        MLGE.M(9,*) = MLGE.M(9,*) + MLGE.M(J1,*);
173:        END;
174:     TOTALS = TOTALS + MLGE.M;
175:     CALL SDC (MILEAGES);
176:     MILEAGE_IND = 0;
177:     MLGE.M = 0;
178:     IF SYSTEM¬='C' | RLG.RT_#¬=94 THEN RETURN;
179:     MLGE.HDG = 'SUBTOTAL INTERSTATE';
180:     MLGE.M = TOTALS;
181:     CALL SDC (MILEAGES);
182:     TOTALS1 = TOTALS;
183:     TOTALS = 0;
184:     END PRINT_LINE;

185: END SURFTYP;
```

SUMMARY-BY-ROUTES --

Member Name	NBR	
Language	PL/I	
Subroutines	PRINTX1	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- SUMMARY-BY-ROUTES output
	ROADLOG	-- Roadlog file
Instruction	1 - 3	"NBR"
	6	"I"/"C"/"P"/"S"/"A" for DATA=INT/ DATA=INT+PRIM/DATA=PRIM/DATA=SEC/ DATA=ALL
	40 - 43	Beginning route number
	56 - 59	Ending route number

SUMMARY-BY-ROUTES provides a summary by route number and by location code. One line is printed for each route in the route system(s) specified, showing the total route, constructed, unimproved, wye, municipal, county, national forest, Indian reservation, game reserve, state forest, national park, state park, national monument, and military reservation mileage for the route. After all the routes in a system have been printed, the total values for the system are printed. If more than one system is processed (DATA=ALL or DATA= INT+PRIM is specified), a grand total is printed after all the systems have been processed. All Roadlog mileage records are utilized in producing the summary. Mileage which falls under two locations (for example, municipal national forest mileage) is shown in both categories. "EN" records are utilized for detection of the end of a route, causing a line to be printed. All records other than "EN" records and mileage records are bypassed. Three arrays are used for calculating values. The array M is used for calculating route totals. The array M_SYS_TOT is used for system totals; M is added to M_SYS_TOT each time the end of a route is detected before re-initializing M to zero. The array M_TOT is used for calculating grand totals when DATA= INT+PRIM or DATA=ALL is specified; M_SYS_TOT is added to M_TOT after printing the system totals prior to re-initialization of M_SYS_TOT. If DATA=ALL or DATA=SEC is specified, the end-of-file condition will be raised on the Roadlog file when attempting to read past the "EN" record of the last Secondary route. In this case, the end-of-file condition is raised and the character "9" is

placed into the route system code field of the Roadlog record, causing the program to think that a new route system has been encountered. The program will print the system totals, and the grand totals if required. Since "9" has a value larger than "S," the Roadlog key will be larger than "S999," and the program will terminate normally.

The NBR program listing follows:

```
/* :SUMMARY-BY-ROUTES,REPORT=ROADLOG,DATA=XXXXX */

 1: /* :SUMMARY-BY-ROUTES,REPORT=ROADLOG,DATA=XXXXX */

 2: /* VALID DATA PARAMETERS:
 3:      DATA=INT   (STARTKEY=I015,ENDKEY=I999,SYSTEM='I')
 4:      DATA=INT+PRIM   (STARTKEY=I015,ENDKEY=P999,SYSTEM='C')
 5:      DATA=PRIM    (STARTKEY=P001,ENDKEY=P999,SYSTEM='P')
 6:      DATA=SEC     (STARTKEY=S201,ENDKEY=S999,SYSTEM='S')
 7:      DATA=ALL     (STARTKEY=I015,ENDKEY=S999,SYSTEM='A')

 8:     DD STATEMENTS UTILIZED:
 9:       SYSPRINT
10:       PRINTER
11:       ROADLOG
12:                                */

13: SUMRT:  PROCEDURE (PARM) OPTIONS (MAIN);

14: /* INSTRUCTION AND PRINT SUBROUTINE */
15: DECLARE
16:    PARM CHAR(100),
17:   (PRINTER,HEADING(9)) CHAR(132) EXT,
18:    PRINTX ENTRY (PIC'Z'),
19:    SYSTEM CHAR(1) DEF PARM POS(6),
20:    STARTKEY CHAR(13) DEF PARM POS(40),
21:    ENDKEY CHAR(13) DEF PARM POS(56),
22:    #_HDGS PIC'Z' DEF INSTR POS(72),
23:    INSTR CHAR(80) EXT;

24: /* STORAGE OF MILEAGES */
25: DECLARE
26:   (M(14),M_SYS_TOT(14),M_TOT(14)) DEC FIXED (9,3),
27:    1  STRUCT BASED (PTR_STRCT),
28:       2 (D1,D2,D3,D4,D5,D6,D7,D8,D9,D10,D11,D12,D13,D14)
29:          DEC FIXED (9,3),
30:    1  OUT DEF STRING_OUT,
31:       3  RT_# PIC'ZZZZBB',
32:     2  MILEAGE,
33:       3  NET_ROUTE PIC'ZZZZZV.ZZZBBBB',
34:       3  CONSTR PIC'ZZZZZV.ZZZ',
35:       3  UNIMP PIC'ZZZZZV.ZZZ',
36:       3  WYE PIC'ZZV.ZZZBB',
37:       3  CITY PIC'ZZZZV.ZZZ',
38:       3  COUNTY PIC'ZZZZZZV.ZZZ',
39:       3  NFOR PIC'ZZZZZV.ZZZ',
40:       3  IRES PIC'ZZZZZV.ZZZ',
41:       3  GAME PIC'ZZZZV.ZZZ',
42:       3  SFOR PIC'ZZZZV.ZZZ',
43:       3  NPRK PIC'ZZZZV.ZZZ',
44:       3  SPRK PIC'ZZZZV.ZZZ',
45:       3  NMON PIC'ZZZZV.ZZZ',
46:       3  MRES PIC'ZZZZV.ZZZ',
47:    OUT_DESCR CHAR(6) DEF STRING_OUT,
48:    STRING_OUT CHAR(132) STATIC;

49: /* ROADLOG FILE */
50: DECLARE
51:    1  RLG BASED (PTR_RLG),
52:       3  DUM1 CHAR(1),
```

```
/* :SUMMARY-BY-ROUTES,REPORT=ROADLOG,DATA=XXXXX */

 53:        3  SYSTEM CHAR(1),
 54:        3  RT_# PIC'ZZZ',
 55:      3 DUM2 CHAR(9),
 56:        3  REMARK CHAR(2),
 57:      3 (SECTN,ROUTE,CONST,UNIMP)DEC FIXED(5,3),
 58:        3 WYE DEC FIXED(3,3),
 59:        3 DUM3 CHAR(61),
 60:        3 LOCN(2) DEC FIXED(3,0),
 61:    1  RLG2 BASED (PTR_RLG),
 62:        3  DUM1 CHAR(1),
 63:        3  KEY CHAR(13),
 64:    ROADLOG FILE RECORD KEYED INPUT SEQL ENV (INDEXED);

 65: /* OTHER VARIABLES */
 66: DECLARE
 67:    B(5) DEC FIXED (3,0) STATIC INIT (35,23,24,36,38),
 68:    BLANKS CHAR(40) STATIC INIT (' '),
 69:    CARRIAGE PIC'Z',
 70:    CARRIAGE1 CHAR(1) DEF CARRIAGE,
 71:    LOCN_CODE(10) DEC FIXED (3,0) STATIC INIT
 72:      (1,2,3,4,5,9,8,10,7,6),
 73:    SAVE_SYS CHAR(1),
 74:    SYS(5) CHAR(50) STATIC INIT (
 75:       ' -- FEDERAL AID INTERSTATE SYSTEM',
 76:       ' -- FEDERAL AID PRIMARY SYSTEM (LESS INTERSTATE)',
 77:       ' -- FEDERAL AID PRIMARY AND INTERSTATE SYSTEMS',
 78:       ' -- FEDERAL AID SECONDARY SYSTEM',
 79:       ' -- ALL FEDERAL AID SYSTEMS');


 80: /***** INITIALIZATION *****/

 81:    CALL INIT (PARM);
 82:    #_HDGS = 8;
 83:    IF SYSTEM='I' THEN I = 1;
 84:       ELSE IF SYSTEM='P' THEN I = 2;
 85:       ELSE IF SYSTEM='C' THEN I = 3;
 86:       ELSE IF SYSTEM='S' THEN I = 4;
 87:       ELSE IF SYSTEM='A' THEN I = 5;
 88:       ELSE DO;
 89:          PRINTER = '*** INVALID DATA SPECIFICATION';
 90:          CALL PRINTX (9);
 91:          GOTO RETURN;
 92:          END;
 93:    HEADING(1) = SUBSTR(BLANKS,1,B(I)) || 'DETAIL SUMMARY BY ROUTES' ||
 94:       SYS(I);
 95:    HEADING(5) = '           NET                                    ' ||
 96:       '********************************** LOCATED IN ' ||
 97:       '**********************************';
 98:    HEADING(6) = 'ROUTE    ROUTE      ** DEVELOPMENT STATUS **     ' ||
 99:       '                  NATIONAL  INDIAN    GAME      ' ||
100:       'STATE  NATL  STATE   NATL  MILIT';
101:    HEADING(7) = ' NO.    MILEAGE      CONSTR   UNIMP  WYE        ' ||
102:       '  CITY    COUNTY   FOREST  RESERVE   REFUGE  F' ||
103:       'OREST PARKS  PARKS    MON RESERVE';

104:    /* INIT VAR */
105:    M, M_SYS_TOT, M_TOT = 0;
```

```
/* :SUMMARY-BY-ROUTES,REPORT=ROADLOG,DATA=XXXXX */

106:     CARRIAGE1 = '1';
107:     STRING_OUT = ' ';

108:     /* INIT ROADLOG FILE */
109:     OPEN FILE (ROADLOG);
110:     ON ENDFILE (ROADLOG) RLG.SYSTEM = '9';
111:     READ FILE (ROADLOG) SET (PTR_RLG) KEY (STARTKEY);


112: /***** EXECUTION LOOP *****/

113: LOOP:
114:     IF RLG.REMARK='EN' THEN GOTO END_OF_ROUTE;
115:     IF RLG.REMARK='  ' | RLG.REMARK='SP' | RLG.REMARK='LP' |
116:        RLG.REMARK='NE' | RLG.REMARK='OS'
117:        THEN GOTO MILEAGE_RECORD;

118: READ_ROADLOG_RECORD:
119:     READ FILE (ROADLOG) SET (PTR_RLG);
120:     GOTO LOOP;

121: MILEAGE_RECORD:
122:     M(1) = M(1) + RLG.ROUTE;
123:     M(2) = M(2) + RLG.CONST;
124:     M(3) = M(3) + RLG.UNIMP;
125:     M(4) = M(4) + RLG.WYE;
126:     DO I=1 TO 2;
127:        DO J=1 TO 10;
128:           IF RLG.LOCN(I)=LOCN_CODE(J) THEN M(J+4) = M(J+4) + RLG.SECTN
129:           END;
130:        END;
131:     GOTO READ_ROADLOG_RECORD;

132: END_OF_ROUTE:

133:     /* PRINT M */
134:     OUT.RT_# = RLG.RT_#;
135:     PTR_STRCT = ADDR(M);
136:     OUT.MILEAGE = STRUCT;
137:     PRINTER = STRING_OUT;
138:     CALL PRINTX (CARRIAGE);
139:     CARRIAGE1 = '1';

140:     /* ADD ROUTE TOTALS TO SYSTEM TOTALS */
141:     M_SYS_TOT = M_SYS_TOT + M;
142:     M = 0;

143:     /* END OF ROUTE SYSTEM? */
144:     SAVE_SYS = RLG.SYSTEM;
145:     READ FILE (ROADLOG) SET (PTR_RLG);
146:     IF RLG.SYSTEM=SAVE_SYS THEN GOTO LOOP;

147:     /* ADD SYSTEM TOTALS TO GRAND TOTALS */
148:     M_TOT = M_TOT + M_SYS_TOT;

149:     /* PRINT M_SYS_TOT */
150:     IF M_TOT(1)¬=M_SYS_TOT(1) | RLG2.KEY<=ENDKEY THEN DO;
151:        PRINTER = 'SUBTOTAL';
```

```
/* :SUMMARY-BY-ROUTES,REPORT=ROADLOG,DATA=XXXXX */

152:        CALL PRINTX (2);
153:        OUT_DESCR = ' ';
154:        CARRIAGE1 = '1';
155:        END;
156:     ELSE DO;
157:        OUT_DESCR = 'TOTAL';
158:        CARRIAGE1 = '2';
159:        END;
160:     PTR_STRCT = ADDR(M_SYS_TOT);
161: PRINT:
162:     OUT.MILEAGE = STRUCT;
163:     PRINTER = STRING_OUT;
164:     CALL PRINTX (CARRIAGE);
165:     CARRIAGE = 3;

166:     /* MORE PROCESSING REQUIRED? */
167:     IF RLG2.KEY<=ENDKEY THEN DO;
168:        M_SYS_TOT = 0;
169:        GOTO READ_ROADLOG_RECORD;
170:        END;

171:     /* MORE THAN ONE SYSTEM PROCESSED? */
172:     IF OUT_DESCR¬='TOTAL' THEN DO;
173:        OUT_DESCR = 'TOTAL';
174:        PTR_STRCT = ADDR(M_TOT);
175:        GOTO PRINT;
176:        END;
177:     CLOSE FILE (ROADLOG);

178: RETURN:
179:     CALL EXIT (PARM);

180: END SUMRT;
```

SUMMARY-BY-LOCATION -- SUMMARY-BY-LOCATION consists of three separate programs: one for DATA=INT, one for DATA=INT+PRIM, and one for DATA=SEC. The member names for the three are NCRI, NCRC and NCRS, respectively. The command decoder cannot set up the member name for inclusion in the instruction; the supervisor itself fills in the fourth character from the data parameter.

DATA=INT:

Member Name	NCRI
Language	PL/I
Subroutines	PRINTX1
Files	SYSPRINT -- IBM messages PRINTER -- SUMMARY-BY-LOCATION output ROADLOG -- Roadlog file
Instruction	1 - 4 "NCRI"

NCRI prints the SUMMARY-BY-LOCATION for the Interstate system. This summary is based on route mileage, rather than constructed mileage (as are most of the Roadlog summaries). All mileage records in the Interstate system are used in producing the summary (the program checks for non-zero route length rather than for remark codes). All Interstate mileage falls into five categories: municipal, county, national forest, Indian reservation, and game refuge. Municipal and county mileage is outside federal reservations; the other three are inside. For each route is shown the total values in each of the five categories, as well as the total mileage outside federal reservations, inside federal reservations, and grand total. Totals for a route are accumulated in array STORE in structure NUM1. The 8 subscripts of STORE are used as:

1 Municipal
2 County
3 Total outside Federal reservations
4 National Forest
5 Indian Reservation
6 Game refuge
7 Total inside Federal reservations
8 Grand total

Mileage falling into two categories is shown in the category falling inside a federal reservation. The Interstate system totals are accumulated in array TOTALS in structure NUM2. TOTALS is subscripted identically to STORE.

The NCRI program listing follows:

```
/* :SUMMARY_BY_LOCATION,REPORT=ROADLOG,DATA=INT */

 1: /* :SUMMARY_BY_LOCATION,REPORT=ROADLOG,DATA=INT */
 2: SUMLOC:  PROCEDURE (PARM) OPTIONS (MAIN);
 3: -
 4:          /** FILE FOR ROADLOG INPUT **/
 5:          DCL ROADLOG FILE RECORD INPUT SEQL KEYED ENV(INDEXED);
 6:          /** FOR PRINTING OUTPUT **/
 7:          DCL HEADING(9)              CHAR(132) EXT,
 8:              INSTR CHAR(80) EXT,
 9:              #_HDGS PIC'Z' DEF INSTR POS(72),
10:              F(0:9) PIC'Z' STATIC INIT (0,1,2,3,4,5,6,7,8,9),
11:              PRINTER                 CHAR(132) EXT,
12:               BLANKS CHAR(132) STATIC INIT (' ');
13:          /** FOR INPUT OF ROADLOG VARIABLES **/
14:          DCL 1 RLG STATIC,
15:                5 DUMMY1              CHAR(1),
16:                5 KEY,
17:                  10 SYSTEM           CHAR(1),
18:                  10 RT_#             PIC'999',
19:                  10 MILEPOST CHAR(9),
20:                5 REMARK CHAR(2),
21:                5 (DUM1,ROUTE)DEC FIXED(5,3),
22:                5 DUMMY2 CHAR(69),
23:        5 LOCN(2) DEC FIXED(3,0),
24:        5 DUMMY3 CHAR(25);
25:          /** FOR OUTPUT OF INTERSTATE MILEAGE TOTALS PER ROUTE **/
26:          DCL 1 NUM1 STATIC,
27:                5 RT_#                PIC'ZZZZZZ',
28:                5 STORE(8)            PIC'ZZZZZV.ZZZ';
29:          DCL STRING_NUM1 CHAR(78) DEF NUM1,
30:              STRING_NUM2 CHAR(78) DEF NUM2;
31:          /** FOR OUTPUT OF COLUMN TOTALS **/
32:          DCL 1 NUM2 STATIC,
33:                5 REMARK              CHAR(6) INIT(' TOTAL'),
34:                5 TOTAL(8)            PIC'ZZZZZV.ZZZ';
35:          /** LOCATION CODES USED IN VARIABLE LOCN IN RLG **/
36: DCL LOCATION_CODE(6) DEC FIXED(3,0) STATIC INIT(1,2,0,3,4,5);
37:          /** BEGINNING KEY OF FAI RECORDS **/
38:          DCL KEY CHAR(13) STATIC INIT ('I015000+0.000');

39:          DCL PARM CHAR(100);
40:     CALL INIT (PARM);
41:          /** SET UP PAGE HEADINGS **/
42:          HEADING(1)=SUBSTR(BLANKS,1,37) || 'SUMMARY  OF  ROUTE  ' ||
43:              'LENGTHS  AND  LOCATIONS';
44:          HEADING(2)=SUBSTR(BLANKS,1,43) || 'FEDERAL  AID  INTERSTATE'
45:              '  SYSTEM';
46:          SUBSTR(HEADING(4),21) = '          OUTSIDE FEDERAL RESERV.' ||
47:             '*** INSIDE FEDERAL RESERVATIONS ***';
48:          SUBSTR(HEADING(6),21)='   ROUTE                              ' ||
49:             'NATIONAL   INDIAN    GAME                GRAND';
50:          SUBSTR(HEADING(7),21)='    NO.   MUNIC   COUNTY    TOTAL ' ||
51:             ' FOREST   RESERV.   REFUGE    TOTAL   TOTAL';
52:          #_HDGS=8;
53:
54:          OPEN FILE (ROADLOG) INPUT TITLE ('ROADLOG');
55:          /** INITIALIZE INTERNAL VARIABLES **/
56:          STORE=0;  TOTAL=0;
57:          /** READ FIRST FAI RECORD **/
```

```
/* :SUMMARY_BY_LOCATION,REPORT=ROADLOG,DATA=INT */

 58:          READ FILE (ROADLOG) INTO (RLG) KEY (KEY);
 59:          GO TO CHECK;
 60:
 61:
 62:          /** MAIN EXECUTION LOOP **/
 63:
 64:          /** CONTINUE READING FAI RECORDS **/
 65: CONT:    READ FILE (ROADLOG) INTO (RLG);
 66:          /** AFTER LAST ROUTE, OUTPUT COLUMN TOTALS **/
 67:          IF SYSTEM¬='I' THEN GO TO OUTPUT_2;
 68:          /** OUTPUT MILEAGES AT END OF EACH ROUTE **/
 69:          IF RLG.REMARK='EN' THEN GO TO OUTPUT_1;
 70:          /** IGNORE DESCRIPTION AND COINCIDENT TYPE RECORDS **/
 71: CHECK:   IF RLG.ROUTE¬=0 THEN GOTO SORT;
 72:          GO TO CONT;
 73:
 74:
 75:          /** OUTPUT_1 PRINTS MILEAGE TOTALS FOR A SINGLE ROUTE **/
 76:
 77:          /** CALCULATE NECESSARY SUBTOTALS AND TOTALS **/
 78: OUTPUT_1: STORE(3) = STORE(1) + STORE(2);
 79:          DO I = 4 TO 6;
 80:             STORE(7) = STORE(7) + STORE(I);
 81:             END;
 82:          STORE(8) = STORE(3) + STORE(7);
 83:          /** PRINT ROUTE MILEAGE **/
 84:          NUM1.RT_#=RLG.KEY.RT_#;
 85:          PRINTER = SUBSTR(BLANKS,1,20) || STRING_NUM1;
 86:        CALL PRINTX (F(1));
 87:          /** STORE ROUTE MILEAGES FOR SYSTEM TOTALS **/
 88:          DO I = 1 TO 8;
 89:             TOTAL(I) = TOTAL(I) + STORE(I);
 90:             END;
 91:          /** SET ROUTE MILEAGES TO ZERO AT START OF EACH NEW ROUTE **/
 92:          STORE=0;
 93:          GO TO CONT;
 94:
 95:          /** OUTPUT_2 PRINTS COLUMN TOTALS **/
 96: OUTPUT_2:  PRINTER = SUBSTR(BLANKS,1,20) || STRING_NUM2;
 97:          CALL PRINTX (F(2));
 98:          GO TO CLOSE;
 99:
100:
101:          /** THE SORT ROUTINE DETERMINES THE TYPE OF MILEAGE IN A
102:              PARTICULAR RECORD AND STORES THE MILEAGE IN AN ARRAY **/
103:          /** SINGLE OR DUAL MILEAGE CLASSIFICATION RECORD **/
104: SORT:
105:            IF LOCN(1)¬=0 & LOCN(2)¬=0 THEN GOTO S1;
106:          /** SORT SINGLE CLASSIFICATION RECORDS**/
107:          DO I = 1,2,4 TO 6;
108:             IF LOCN(1) = LOCATION_CODE(I) THEN DO;
109:                /** STORE DETECTED MILEAGE TYPE **/
110:                STORE(I) = STORE(I) + ROUTE;
111:                END;
112:             END;
113:          GOTO CONT;
114:          /** SORT DUAL CLASSIFICATION RECORDS **/
115: S1:      DO I = 4 TO 6;
```

```
/* :SUMMARY_BY_LOCATION,REPORT=ROADLOG,DATA=INT */

116:              IF LOCN(2) = LOCATION_CODE(I) THEN DO;
117:                 STORE(I) = STORE(I) + ROUTE;
118:                 END;
119:              END;
120:           GOTO CONT;
121:
122:
123: CLOSE:  CLOSE FILE (ROADLOG);
124:           CALL EXIT (PARM);

125: END SUMLOC;
```

DATA=INT+PRIM:

Member Name	NCRC	
Language	PL/I	
Subroutines	PRINTX1	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- SUMMARY-BY-LOCATION output
	ROADLOG	-- Roadlog file
Instruction	1 - 4	"NCRC"

NCRC prints the SUMMARY-BY-LOCATION when DATA=INT+PRIM is specified. This summary is based on constructed length, rather than on route length. The summary must also compute and print the status of the 7% system (consisting of Interstate and Primary mileage outside Federal reservations which is neither urban nor Interstate loop mileage). Mileage in the Interstate and Primary systems falls into seven categories: municipal, county, National Forest, Indian Reservation, state forest, National park, and game refuge mileage. Municipal, county, and state forest mileage is outside Federal reservations. Values for each route are accumulated in 12-element array STORE in structure NUM1. The subscript usage is:

1	Municipal
2	County
3	State Forest
4	Totals -- 1 + 2 + 3
5	Urban Extensions
6	Totals -- 4 - 5
7	National Forest
8	Indian Reservation
9	National Parks
10	Game Refuge
11	Totals -- 7 + 8 + 9 + 10
12	Totals -- 4 + 11

Out-of-state mileage records (remark "OS") are not included in the summary. Mileage falling into two categories is included in the category falling inside a federal reservation (thus

removing it from the 7% system). "IL" records in the Primary system defining Interstate loops are processed to find non-urban Interstate loop mileage. All such mileage is added into variable LOOP_MI for use when calculating the 7% system status. Array TOTAL in structure NUM3 is used for calculating system totals. Like STORE, TOTAL has 12 subscripts, used as in STORE. The 7% system is calculated by taking the total mileage (column 4 of the summary) outside federal reservations, and subtracting the urban extensions (column 5) and non-urban Interstate loops (in LOOP_MI). The result is compared to the permissible value of 4697.0 miles, and any overrun or underrun printed.

The NCRC program listing follows:

```
/* :SUMMARY_BY_LOCATION,REPORT=ROADLOG,DATA=INT+PRIM */

  1: /* :SUMMARY_BY_LOCATION,REPORT=ROADLOG,DATA=INT+PRIM */

  2: SUMLOC:  PROCEDURE (PARM) OPTIONS (MAIN);
  3:
  4:
  5:         /** FILE FOR ROADLOG DATA INPUT **/
  6:         DCL ROADLOG FILE RECORD INPUT SEQL KEYED ENV(INDEXED);
  7:         /** FOR PRINTING OUTPUT **/
  8:         DCL HEADING(9)              CHAR(132) EXT,
  9:             INSTR CHAR(80) EXT,
 10:             PARM CHAR(100),
 11:             #_HDGS PIC'Z' DEF INSTR POS(72),
 12:             #_LINES                 PIC'ZZ',
 13:             F(0:9) PIC'Z' STATIC INIT (0,1,2,3,4,5,6,7,8,9),
 14:             PRINTER                 CHAR(132) EXT,
 15:             BLANKS                  CHAR(132) INIT(' ');
 16:         /** FOR INPUT OF ROADLOG VARIABLES **/
 17:         DCL 1 RLG STATIC,
 18:               5 DUMMY1              CHAR(1),
 19:               5 KEY,
 20:                 10 SYSTEM           CHAR(1),
 21:                 10 RT_#             PIC'999',
 22:                 10 MILEPOST CHAR(9),
 23:             5 REMARK CHAR(2),
 24:             5 SECTN DEC FIXED(5,3),
 25:             5 DUMMY2 CHAR(9),
 26:             5 WYE DEC FIXED(3,3),
 27:             5 DESCR CHAR(35),
 28:             5 DUMMY3 CHAR(13),
 29:             5 POPULATION_CODE DEC FIXED(1,0),
 30:             5 DUMMY4 CHAR(12),
 31:             5 LOCN(2) DEC FIXED(3,0),
 32:     5 DUMMY5 CHAR(25);
 33:         /** FOR OUTPUT OF MILEAGE TOTALS PER ROUTE **/
 34:         DCL 1 NUM1 STATIC,
 35:               5 DUMMY1              CHAR(13) INIT(' '),
 36:               5 RT_#                PIC'ZZ',
 37:               5 STORE(12)           PIC'ZZZZZV.ZZZ';
 38:         /** FOR OUTPUT OF COLUMN TOTALS **/
 39:         DCL 1 NUM3 STATIC,
 40:               5 REMARK              CHAR(15) INIT('     INTERSTATE'),
 41:               5 TOTAL(12)           PIC'ZZZZZV.ZZZ';
 42:         /** FOR DETERMINING NET 7% SYSTEM MILEAGE **/
 43:         DCL 1 NUM2 STATIC,
 44:               5 REMARK(2)           CHAR(42),
 45:               5 MILEAGE             PIC'ZZZZZZV.ZZZ';
 46:         /** FOR 7% SYSTEM SUMMARY OUTPUT **/
 47:         DCL 1 NUM4 STATIC,
 48:               5 REMARK(2)           CHAR(42),
 49:               5 MILEAGE             PIC'ZZZZZZV.Z';
 50:         /** FOR ACCUMULATING TOTAL SYSTEM AND LOOP MILEAGES **/
 51:         DCL LOOP_MI                 PIC'ZZZVZZZ',
 52:             TOTAL_MI(12)            PIC'ZZZZZVZZZ';
 53:         /** LOCATION CODES USED IN VARIABLE LOCN IN RLG **/
 54:       DCL LOCATION_CODE(10) DEC FIXED(3,0) STATIC INIT
 55:           (1,2,9,0,0,0,3,4,8,5);
 56:         /** KEYS FOR INTERSTATE RECORD SEARCHING **/
 57:         DCL SAVE_KEY CHAR(13),
```

```
/* :SUMMARY_BY_LOCATION,REPORT=ROADLOG,DATA=INT+PRIM */

 58:             LOOP_ENDKEY CHAR(13);
 59:
 60:          /** PROGRAM INITIALIZATION **/

 61:     ON ERROR BEGIN;
 62:        PRINTER = '*** ERROR IN SUMMARY_BY_LOCATION ROUTINE';
 63:        CALL PRINTX (F(3));
 64:        GOTO CLOSE;
 65:        END;
 66:     CALL INIT (PARM);
 67:          /** SET UP PAGE HEADINGS **/
 68:       HEADING(1)=SUBSTR(BLANKS,1,40)||'SUMMARY  OF  ROUTE  LE' ||
 69:             'NGTHS  AND  LOCATIONS';
 70:          SUBSTR(HEADING(2),41,44) =
 71:             'FEDERAL AID PRIMARY AND INTERSTATE SYSTEMS';
 72:          HEADING(4) = '                        ******LOCATED OUTSIDE FEDE'
 73:             'RAL RESERVATIONS******      **LOCATED INSIDE FEDERAL R'
 74:             'ESERVATIONS**';
 75:       HEADING(6)=SUBSTR(BLANKS,1,54)||'LESS';
 76:          HEADING(7) = '             ROUTE                          STATE   '
 77:           '  SUB    URBAN                NAT.    INDIAN     NAT.   '||
 78:             ' GAME              GRAND';
 79:          HEADING(8) = '               NO.   MUNIC   COUNTY   FOREST   '
 80:             ' TOTAL  EXTENT.    TOTAL   FOREST  RESERV.    PARKS   '
 81:             'REFUGE    TOTAL   TOTAL';
 82:          #_HDGS = 9;
 83:
 84:          OPEN FILE (ROADLOG) INPUT TITLE ('ROADLOG');
 85:          /** INITIALIZE INTERNAL VARIABLES **/
 86:          TOTAL_MI=0;  TOTAL,STORE=0;
 87:          LOOP_MI=0;
 88:          /** READ FIRST RECORD **/
 89:     READ FILE (ROADLOG) INTO (RLG) KEY ('I015');
 90:          GO TO CHECK1;
 91:
 92:
 93:          /** MAIN EXECUTION LOOP **/
 94:
 95:          /** CONTINUE READING RECORDS **/
 96: CONT1:  READ FILE (ROADLOG) INTO (RLG);
 97:          /** INTERSTATE LOOP MILEAGE? **/
 98:          IF RLG.REMARK='IL' THEN GO TO INTERSTATE_LOOPS;
 99:          /** OUTPUT TOTALS AT END OF EACH ROUTE **/
100:          IF RLG.REMARK='EN' THEN GO TO OUTPUT1;
101:          /** AFTER LAST PRIMARY ROUTE, OUTPUT TOTALS AND 7% STATUS **/
102:          IF RLG.KEY.SYSTEM='S' THEN GO TO OUTPUT2;
103:          /** IGNORE DESCRIPTION AND COINCIDENT TYPE RECORDS **/
104: CHECK1: IF RLG.REMARK='  ' | RLG.REMARK='SP' | RLG.REMARK='LP' |
105:             RLG.REMARK='NE' THEN CALL SORT;
106:          GO TO CONT1;
107:
108:          /** SEARCHES FOR AND CUMULATES INTERSTATE LOOP MILEAGE **/
109: INTERSTATE_LOOPS:
110:          /** RETAIN CURRENT PRIMARY SYSTEM KEY **/
111:          SAVE_KEY=STRING(RLG.KEY);
112:          /** DETERMINE ENDING KEY FOR LOOP RECORDS **/
113:          LOOP_ENDKEY = SUBSTR(RLG.DESCR,6,4) || SUBSTR(RLG.DESCR,26,9)
114:          /** BEGIN READING LOOP RECORDS **/
```

```
/* :SUMMARY_BY_LOCATION,REPORT=ROADLOG,DATA=INT+PRIM */

115:          READ FILE (ROADLOG) INTO (RLG) KEY (SUBSTR(RLG.DESCR,6,4) ||
116:             SUBSTR(RLG.DESCR,16,9));
117: CHECK2:  IF RLG.REMARK¬='   ' & RLG.REMARK¬='SP' & RLG.REMARK¬='LP' &
118:             RLG.REMARK¬='OS' & RLG.REMARK¬='NE' THEN GO TO CONT2;
119:          /** MILEAGE OUTSIDE FEDERAL RESERVATIONS? **/
120:          IF LOCN(1)=1 & LOCN(2)=0 | LOCN(1)=1 & LOCN(2)=9 |
121:             LOCN(1)=2 & LOCN(2)=0 | LOCN(1)=9 & LOCN(2)=0 THEN DO;
122:                /** MILEAGE NOT ALREADY COUNTED AS AN URBAN EXTENSION? */
123:                IF POPULATION_CODE < 4 THEN DO;
124:                   LOOP_MI = LOOP_MI + SECTN;
125:                   END;
126:                END;
127:          /** CONTINUE READING LOOP MILEAGE RECORDS **/
128: CONT2:   READ FILE (ROADLOG) INTO (RLG);
129:          IF LOOP_ENDKEY > STRING(RLG.KEY) THEN GO TO CHECK2;
130:          /** RETURN TO PRIMARY SYSTEM RECORDS **/
131: RETURN:  READ FILE (ROADLOG) INTO (RLG) KEY (SAVE_KEY);
132:          GO TO CONT1;
133:
134:
135:          /** OUTPUT1 PRINTS TOTALS PER ROUTE AND SYSTEM SUBTOTALS **/
136: OUTPUT1:
137:          STORE(4) = STORE(1) + STORE(2) + STORE(3);
138:          STORE(6) = STORE(4) - STORE(5);
139:          STORE(11) = STORE(7) + STORE(8) + STORE(9) + STORE(10);
140:          STORE(12) = STORE(4) + STORE(11);
141:          /** PRINT ROUTE MILEAGE **/
142:          NUM1.RT_# = RLG.KEY.RT_#;
143:           PRINTER=STRING(NUM1);
144:              CALL PRINTX (F(1));
145:          /** STORE ROUTE MILEAGES FOR SYSTEM SUBTOTALS **/
146:          DO I = 1 TO 12;
147:             TOTAL(I) = TOTAL(I) + STORE(I);
148:             END;
149:          /** PRINT SUBTOTALS WHEN NECESSARY **/
150:          IF RLG.KEY.RT_# = 094 | RLG.KEY.RT_# = 054 THEN DO;
151:             PRINTER = '       SUBTOTAL';
152:          CALL PRINTX (F(1));
153:             IF RLG.KEY.SYSTEM = 'P' THEN NUM3.REMARK ='        PRIMARY';
154:             PRINTER = STRING(NUM3);
155:          CALL PRINTX (F(1));
156:             PRINTER = ' ';
157:          CALL PRINTX (F(1));
158:             /** STORE MILEAGES FOR 7% SYSTEM TOTALS **/
159:             DO I = 1 TO 12;
160:                TOTAL_MI(I) = TOTAL_MI(I) + TOTAL(I);
161:                END;
162:             TOTAL=0;
163:             END;
164:          STORE=0;
165:          GO TO CONT1;
166:
167:
168:          /** OUTPUT2 PRINTS TOTALS AND STATUS OF THE 7% SYSTEM **/
169: OUTPUT2:
170:          PRINTER = 'NET CONSTRUCTED';
171:          CALL PRINTX (F(1));
172:          NUM3.REMARK = '         LENGTH';
```

```
/* :SUMMARY_BY_LOCATION,REPORT=ROADLOG,DATA=INT+PRIM */

173:          TOTAL = TOTAL_MI;
174:          PRINTER = STRING(NUM3);
175:          CALL PRINTX (F(1));
176:
177:          /** OUTPUTS STATUS OF 7% SYSTEM MILEAGE **/
178:          HEADING(4)=' ';  #_HDGS=5;  #_LINES=15;
179:          PRINTER=SUBSTR(BLANKS,1,38) || '*****************************'
180:              '**********************';
181:          CALL PRINTXA (F(8),#_LINES);
182:          PRINTER=SUBSTR(BLANKS,1,34) || '********* STATUS  OF  THE  '
183:              '7%  SYSTEM  MILEAGE *********';
184:          CALL PRINTX (F(1));
185:          PRINTER=SUBSTR(BLANKS,1,38) || '*****************************'
186:              '**********************';
187:          CALL PRINTX (F(1));
188:          NUM2.REMARK(1) = '                                          TOTAL 7% S'
189:          NUM2.REMARK(2) = 'YSTEM MILEAGE OUTSIDE FEDERAL RESERVATIONS';
190:          NUM2.MILEAGE = TOTAL(4);
191:           PRINTER = STRING(NUM2);
192:          CALL PRINTX (F(3));
193:          NUM2.REMARK(1) = ' ';
194:          NUM2.REMARK(2) = '                LESS URBAN EXTENSION MILEAGE'
195:          NUM2.MILEAGE = TOTAL(5);
196:           PRINTER = STRING(NUM2);
197:          CALL PRINTX (F(1));
198:          NUM2.REMARK(2) = '   LESS NON-URBAN INTERSTATE LOOP MILEAGE';
199:          NUM2.MILEAGE = LOOP_MI;
200:           PRINTER = STRING(NUM2);
201:          CALL PRINTX (F(1));
202:          NUM2.REMARK(1) = '                                        ACTUAL 7% S'
203:          NUM2.REMARK(2) = 'YSTEM MILEAGE OUTSIDE FEDERAL RESERVATIONS';
204:          NUM2.MILEAGE = TOTAL(4) - TOTAL(5) - LOOP_MI;
205:           PRINTER = STRING(NUM2);
206:          CALL PRINTX (F(1));
207:          NUM4.REMARK(1) = '                                   PERMISSIBLE 7% S'
208:          NUM4.REMARK(2) = 'YSTEM MILEAGE OUTSIDE FEDERAL RESERVATIONS'
209:          NUM4.MILEAGE = 4697.0;
210:           PRINTER = STRING(NUM4);
211:          CALL PRINTX (F(3));
212:          NUM4.REMARK(1) = '                                        ACTUAL 7% S'
213:          NUM4.MILEAGE = ROUND(NUM2.MILEAGE,1);
214:           PRINTER = STRING(NUM4);
215:          CALL PRINTX (F(1));
216:          NUM4.REMARK(1) = ' ';
217:          IF NUM4.MILEAGE>=4697.0 THEN DO;
218:             NUM4.REMARK(2)='       OVERRUN IN ACTUAL 7% SYSTEM MILEAGE
219:             NUM4.MILEAGE = ROUND(NUM2.MILEAGE,1) - 4697.0;
220:             END;
221:           ELSE DO;
222:             NUM4.REMARK(2)='      UNDERRUN IN ACTUAL 7% SYSTEM MILEAGE
223:             NUM4.MILEAGE = 4697.0 - ROUND(NUM2.MILEAGE,1);
224:             END;
225:          PRINTER = STRING(NUM4);
226:          CALL PRINTX (F(1));
227:          GO TO CLOSE;
228:
229:          /** THE SORT ROUTINES DETERMINE THE TYPE OF MILEAGE IN A
230:              PARTICULAR RECORD AND STORE THE MILEAGES IN ARRAYS **/
```

```
/* :SUMMARY_BY_LOCATION,REPORT=ROADLOG,DATA=INT+PRIM */

231: SORT:    PROC;
232:          /** SINGLE OR DUAL MILEAGE CLASSIFICATION RECORD **/
233:          IF LOCN(1)¬=0 & LOCN(2)¬=0 THEN GOTO S1;
234:          /** SORT SINGLE CLASSIFICATION RECORDS **/
235:          DO I = 1 TO 3,7 TO 10;
236:             IF LOCN(1) = LOCATION_CODE(I) THEN DO;
237:                /** STORE DETECTED MILEAGE TYPE **/
238:                STORE(I) = STORE(I) + SECTN;
239:                END;
240:             END;
241:          GO TO S2;
242:          /** SORT DUAL CLASSIFICATION RECORDS **/
243: S1:      DO I = 3,7 TO 10;
244:             IF LOCN(2) = LOCATION_CODE(I) THEN DO;
245:                STORE(I) = STORE(I) + SECTN;
246:                END;
247:             END;
248:          /** CHECK ALL RECORDS FOR URBAN EXTENSIONS **/
249: S2:   IF LOCN(1)=1 &(LOCN(2)=0|LOCN(2)=9)|LOCN(1)=2 &
250:             LOCN(2)=0  | LOCN(1)=9 & LOCN(2)=0 THEN DO;
251:                IF POPULATION_CODE>=4 THEN DO;
252:                   STORE(5) = STORE(5) + SECTN;
253:                   END;
254:             END;
255:          END SORT;
256:
257:
258: CLOSE:   CLOSE FILE (ROADLOG);
259:          CALL EXIT (PARM);

260: END SUMLOC;
```

DATA=SEC:

Member Name	NCRS	
Language	PL/I	
Subroutines	PRINTX1	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- SUMMARY-BY-LOCATION output
	ROADLOG	-- Roadlog file
Instruction	1 - 4	"NCRS"

NCRS operates in a similar fashion to NCRI, excepting that more categories exist. NCRS uses route mileage in its computations. The subscripts of STORE and TOTAL have the following meanings in NCRS:

1 Municipal
2 County
3 State Forest
4 State Park
5 Total -- outside Federal reservations
6 National Forest
7 Indian Reservations
8 Military Reservations
9 Game refuge
10 National Monument
11 Total -- inside Federal reservations
12 Grand total

The remainder of the discussion of NCRI is applicable to NCRS.

The NCRS program listing follows:

```
/* :SUMMARY_BY_LOCATION,REPORT=ROADLOG,DATA=SEC */

 1: /* :SUMMARY_BY_LOCATION,REPORT=ROADLOG,DATA=SEC */
 2: SUMLOC: PROC(PARM) OPTIONS(MAIN);
 3: /** SUMMARY OF ROUTE LENGTHS BY LOCATION FOR THE FAS SYSTEM **/
 4: /** FILE FOR ROADLOG INPUT **/
 5: DCL ROADLOG FILE RECORD INPUT SEQL KEYED ENV(INDEXED);
 6: /** FOR PRINTING OUTPUT **/
 7: DCL HEADING(9)    CHAR(132) EXT,
 8:   INSTR CHAR(80)EXT,
 9:   #_HDGS PIC'Z' DEF INSTR POS(72),
10:   PARM CHAR(100),
11:   #_LINES PIC'ZZ',
12:   F(0:9) PIC'Z' INIT(0,1,2,3,4,5,6,7,8,9),
13:   PRINTER CHAR(132)EXT,
14:   BLANKS CHAR(132)INIT(' ');
15: /** FOR INPUT OF ROADLOG VARIABLES **/
16: DCL 1 RLG STATIC,
17:    5 DUMMY1 CHAR(1),
18:     5 KEY,
19:      10 SYSTEM CHAR(1),
20:      10 (RT_#,MILEPOST) PIC'ZZZ',
21:      10 OFFSET CHAR(6),
22:     5 REMARK CHAR(2),
23:     5 DUMMY2 CHAR(3),
24:     5 ROUTE DEC FIXED(5,3),
25:     5 DUMMY3 CHAR(8),
26:     5 DESCR CHAR(35),
27:     5 DUMMY4 CHAR(26),
28:     5 LOCN(2) DEC FIXED(3,0),
29:     5 DUMMY5 CHAR(25);
30: /** FOR OUTPUT OF SECONDARY MILEAGE TOTALS PER ROUTE **/
31: DCL 1 NUM1 STATIC,
32:     5 RT_# PIC'ZZZZZ',
33:     5 STORE(12) PIC'ZZZZZV.ZZZ';
34: /** FOR OUTPUT OF COLUMN TOTALS **/
35: DCL 1 NUM2 STATIC,
36:     5 REMARK CHAR(5) INIT('TOTAL'),
37:     5 TOTAL(12) PIC'ZZZZZV.ZZZ';
38: /** LOCATION CODES USED IN VARIABLE LOCN IN RLG **/
39:  DCL LOCATION_CODE(10) DEC FIXED(3,0)
40:    INIT(1,2,9,10,0,3,4,6,5,7);
41: /** BEGINNING KEY OF FAS RECORDS **/
42:  DCL KEY CHAR(16) STATIC INIT('S201000+0.000');
43:  ON ERROR BEGIN;
44:    PRINTER='***ERROR IN SUMMARY_BY_LOCATION ROUTINE';
45:    CALL PRINTX(F(3));
46:    GOTO CLOSE;
47:    END;
48:  CALL INIT(PARM);
49: /* SET PAGE HEADINGS */
50:  HEADING(1)=SUBSTR(BLANKS,1,35)||'SUMMARY  OF  ROUTE  ' ||
51:    'LENGTHS  AND  LOCATIONS';
52:  HEADING(2)=SUBSTR(BLANKS,1,41)||'FEDERAL  AID  '||
53:    'SECONDARY  SYSTEM';
54:  HEADING(4)='          LOCATED  OUTSIDE  FEDERAL  RESERVATIONS'||
55:    '    LOCATED      INSIDE      FEDERAL      RESERVATIONS';
56:  HEADING(6)=' ROUTE                          STATE      STATE      '||
57:     '    NATIONAL INDIAN   MILITARY  GAME   NATIONAL      '||
58:    '    COMBINED';
```

```
/* :SUMMARY_BY_LOCATION,REPORT=ROADLOG,DATA=SEC */

 59:  HEADING(7)='  NO.    MUNIC   COUNTY   FOREST     PARK     ' ||
 60:    'TOTAL  FOREST   RESERV.  RESERV.   REFUGE MONUMENT    '||
 61:    'TOTAL   TOTAL';
 62:  #_HDGS=8;
 63:  OPEN FILE(ROADLOG) INPUT TITLE('ROADLOG');
 64: /** INITIALIZE INTERNAL VARIABLES */
 65:  STORE=0;  TOTAL =0;
 66:  ON ENDFILE(ROADLOG) GOTO OUTPUT_2;
 67: /** READ FIRST FAS RECORD **/
 68:  READ FILE(ROADLOG) INTO(RLG) KEY(KEY);
 69:  GOTO CHECK;
 70:  /*  MAIN EXECUTION LOOP **/
 71: /* CONTINUE READING FAS RECORDS */
 72: CONT:  READ FILE (ROADLOG) INTO (RLG);
 73: /** AFTER LAST ROUTE, OUTPUT SYSTEM TOTALS **/
 74:         IF RLG.KEY.SYSTEM¬='S' THEN GOTO OUTPUT_2;
 75: /* OUTPUT MILEAGES AT END OF EACH ROUTE */
 76:  IF RLG.REMARK='EN' THEN GOTO OUTPUT_1;
 77: /* IGNORE DESCRIPTION AND COINCIDENT TYPE RECORDS **/
 78: CHECK: IF RLG.REMARK=' '|RLG.REMARK='SP'|RLG.REMARK='LP'|
 79:     RLG.REMARK='NE'|RLG.REMARK='OS'  THEN CALL SORT;
 80:  GOTO CONT;
 81: /** OUTPUT_1 PRINTS MILEAGE TOTALS FOR A SINGLE ROUTE */
 82: /* CALCULATE NECESSARY SUBTOTALS AND TOTALS */
 83: OUTPUT_1: STORE(5)=STORE(1)+STORE(2)+STORE(3)+STORE(4);
 84:   DO I=6 TO 10;
 85:     STORE(11)=STORE(11)+STORE(I);
 86:    END;
 87:   STORE(12)=STORE(5)+STORE(11);
 88: /* PRINT ROUTE MILEAGE */
 89:  NUM1.RT_#=RLG.RT_#;
 90:    PRINTER=STRING(NUM1);
 91:   CALL PRINTX(F(1));
 92: /* STORE ROUTE MILEAGES FOR SYSTEM TOTALS */
 93:   DO I=1 TO 12;
 94:    TOTAL(I)=TOTAL(I)+STORE(I);
 95:     END;
 96: /* SET ROUTE MILEAGES TO ZERO AT START OF EACH NEW ROUTE */
 97:   STORE=0;
 98:   GOTO CONT;
 99: /* OUTPUT_2 PRINTS COLUMN TOTALS */
100: OUTPUT_2:  PRINTER=STRING(NUM2);
101:     CALL PRINTX(F(2));
102:   GOTO CLOSE;
103: /* THE SORT ROUTINE DETERMINES THE TYPE OF MILEAGE IN A
104:    PARTICULAR RECORD AND STORES THE MILEAGE IN ARRAYS */
105: SORT: PROC;
106: /* SINGLE OR DUAL MILEAGE CLASSIFICATION RECORD */
107:         IF LOCN(1)¬=0 & LOCN(2)¬=0 THEN GOTO S1;
108: /* SORT SINGLE CLASSIFICATION RECORDS */
109:   DO I=1 TO 4,6 TO 10;
110:    IF LOCN(1)=LOCATION_CODE(I) THEN DO;
111: /* STORE DETECTED MILEAGE TYPE */
112:   STORE(I)=STORE(I)+ROUTE;
113:    END;
114:    END;
115:   GOTO S3;
116: /* SORT DUAL CLASSIFICATION RECORDS **/
```

```
/* :SUMMARY_BY_LOCATION,REPORT=ROADLOG,DATA=SEC */

117: S1: IF LOCN(1)¬=1 THEN GOTO S2;
118:    DO I=3,4,6 TO 10;
119:     IF LOCN(2)=LOCATION_CODE(I) THEN DO;
120:     STORE(I)=STORE(I)+ROUTE;
121:      END;
122:      END;
123:    GOTO S3;
124: S2:    IF LOCN(1)=3 | LOCN(2)=3 THEN DO;
125:         STORE(6)=STORE(6)+ROUTE;
126:      END;
127: S3:    END SORT;
128: CLOSE:  CLOSE FILE(ROADLOG);
129:      CALL EXIT (PARM);
130: END SUMLOC;
```

FORHWY-SUMMARY -- FORHWY-SUMMARY is composed of two separate programs: NERL for production of FHSUMMARY=LOCATION, and NERS for FHSUMMARY=SURF-TYPE.

FHSUMMARY=LOCATION:

Member Name	NERL
Language	PL/I
Subroutines	PRINTX1
Files	SYSPRINT -- IBM messages PRINTER -- FORHWY-SUMMARY output ROADLOG -- Roadlog file
Instruction	1 - 4 "NERL"

NERL processes the entire Roadlog file, searching for records with non-zero forest highway number. Mileages are stored by route and by location code in array MLGES. After the end-of-file has been detected on the Roadlog file, the summary is printed.

The NERL program listing follows:

```
/* :FORHWY_SUMMARY,REPORT=ROADLOG,FHSUMMARY=LOCATION */

  1: /* :FORHWY_SUMMARY,REPORT=ROADLOG,FHSUMMARY=LOCATION */
  2: FORHWY:  PROCEDURE (PARM) OPTIONS (MAIN);


  3: /***** DECLARATIION OF VARIABLES (ALPHABETICAL ORDER) *****/


  4: DECLARE
  5:    #_HDGS PIC'Z' DEF INSTR POS(72),
  6:    BLANKS CHAR(132) INIT(' '),
  7:    CODE(3) CHAR(8) INIT('  PRIM','  SEC',' TOTAL'),
  8:    F(0:9) PIC'Z' STATIC INIT (0,1,2,3,4,5,6,7,8,9),
  9:    HEADING(9) CHAR(132) EXT,
 10:    INSTR CHAR(80) EXT,
 11:    JX(3) BIN FIXED INIT(2,1,2),
 12:    1  MLGE STATIC,
 13:       3  ROUTE_PRIM PIC'ZZZVZZZ',
 14:       3  ROUTE_SEC  PIC'ZZZVZZZ',
 15:       3  CONST PIC'ZZZZVZZZ',
 16:       3  UNIMP PIC'ZZZVZZZ',
 17:       3  WYE PIC'ZVZZZ',
 18:       3  CITY PIC'ZZVZZZ',
 19:       3  CNTY PIC'ZZZVZZZ',
 20:       3  NFOR PIC'ZZZVZZZ',
 21:       3  SFOR PIC'ZZZVZZZ',
 22:       3  IRES PIC'ZVZZZ',
 23:       3  PARK PIC'ZVZZZ',
 24:    1  MLGES(65) STATIC LIKE MLGE,
 25:    1  OUT_STRUCT STATIC,
 26:       3  FORHWY_# PIC'ZZZZZBBB',
 27:       3  MILEAGE,
 28:          5  ROUTE_PRIM PIC'ZZZZZV.ZZZ',
 29:          5  ROUTE_SEC PIC'ZZZZZV.ZZZ',
 30:          5  CONST PIC'ZZZZZZZZV.ZZZ',
 31:          5  UNIMP PIC'ZZZZZV.ZZZ',
 32:          5  WYE PIC'ZZZV.ZZZ',
 33:          5  CITY PIC'ZZZZZZZV.ZZZ',
 34:          5  CNTY PIC'ZZZZZV.ZZZ',
 35:          5  NFOR PIC'ZZZZZV.ZZZ',
 36:          5  SFOR PIC'ZZZZV.ZZZ',
 37:          5  IRES PIC'ZZZV.ZZZ',
 38:          5  PARK PIC'ZZZV.ZZZ',
 39:       3  GRAND_TOTAL PIC'ZZZZZZZZZV.ZZZ',
 40:    PARM CHAR(100),
 41:    PRINTER CHAR(132) EXT,
 42:    1  RLG STATIC,
 43:       3  DUMMY1 CHAR(1),
 44:       3  SYSTEM CHAR(1),
 45:       3  DUMMY2 CHAR(12),
 46:       3  REMARK CHAR(2),
 47:       3  (SECTN,ROUTE,CONST,UNIMP)DEC FIXED(5,3),
 48:        3 WYE DEC FIXED(3,3),
 49:      3 DUMMY3 CHAR(57),
 50:       3  (FORHWY_#,DUM1,LOCN(2))DEC FIXED(3,0),
 51:       3 DUMMY4 CHAR(25),
 52:    D4 CHAR(4),
 53:    ROADLOG FILE RECORD KEYED ENV(INDEXED),
 54:    1  TOTALS(3) LIKE MLGE;
```

```
/* :FORHWY_SUMMARY,REPORT=ROADLOG,FHSUMMARY=LOCATION */


 55: /***** PROGRAM INITIALIZATION *****/

 56:    ON ERROR BEGIN;
 57:       PRINTER = '*** ERROR IN FORHWY_SUMMARY ROUTINE';
 58:       CALL PRINTX (F(3));
 59:       GOTO RETURN;
 60:       END;
 61:    CALL INIT (PARM);

 62:    /*** SET UP PAGE HEADINGS ***/
 63:    #_HDGS = 8;
 64:    HEADING(1) = SUBSTR(BLANKS,1,25) ||
 65:       'M O N T A N A   F O R E S T   H I G H W A Y   S Y S T E M';
 66:    HEADING(3) = SUBSTR(BLANKS,1,34) ||
 67:       'C O N S T R U C T I O N   D E T A I L S';
 68:    HEADING(5) = ' FOREST     ROUTE LENGTH          ' ||
 69:       '  IMPROVEMENT STATUS          ' ||
 70:       '***************** LOCATED IN *****************';
 71:    HEADING(6) = 'HIGHWAY   ******************      ' ||
 72:       '************************      ' ||
 73:       '                NATIONAL   STATE  INDIAN                GRAND';
 74:    HEADING(7) = ' ROUTE    PRIMARY SECONDARY     ' ||
 75:       'CONSTR   UNIMP     WYE       ' ||
 76:       'CITY    COUNTY  FOREST   FOREST RESERVE PARKS        TOTAL';

 77:    /*** OPEN ROADLOG FILE ***/
 78:    OPEN FILE (ROADLOG) INPUT SEQL;
 79:    ON ENDFILE (ROADLOG) GOTO PRINT_SUMMARY;
 80:    MLGES = 0;
 81:    TOTALS = 0;


 82: /***** MAIN CONTROL SECTION ****/

 83:    /*** READ ROADLOG FOREST HIGHWAY DATA ****/

 84: READ_RLG_RECORD:
 85:    READ FILE (ROADLOG) INTO (RLG);
 86:    IF RLG.FORHWY_#=0 THEN GOTO READ_RLG_RECORD;
 87:    MLGE = 0;
 88:    IF RLG.SYSTEM='S' THEN MLGE.ROUTE_SEC = RLG.ROUTE;
 89:       ELSE MLGE.ROUTE_PRIM = RLG.ROUTE;
 90:    MLGE.CONST = RLG.SECTN - RLG.UNIMP - RLG.WYE;
 91:    MLGE.UNIMP = RLG.UNIMP;
 92:    MLGE.WYE = RLG.WYE;
 93:    DO J1=1;
 94:     IF RLG.LOCN(J1)=1 THEN MLGE.CITY=RLG.SECTN;
 95:     ELSE IF RLG.LOCN(J1)=2 THEN MLGE.CNTY=RLG.SECTN;
 96:     ELSE IF RLG.LOCN(J1)=3 THEN MLGE.NFOR=RLG.SECTN;
 97:     ELSE IF RLG.LOCN(J1)=9 THEN MLGE.SFOR=RLG.SECTN;
 98:     ELSE IF RLG.LOCN(J1)=4 THEN MLGE.IRES=RLG.SECTN;
 99:     ELSE IF RLG.LOCN(J1)=8 THEN MLGE.PARK=RLG.SECTN;
100:     ELSE IF RLG.LOCN(J1)=10 THEN MLGE.PARK=RLG.SECTN;
101:          ELSE DO;
102:             D4=' ';
103:             IF RLG.LOCN(J1)=5 THEN D4='IRES';
```

```
/* :FORHWY_SUMMARY,REPORT=ROADLOG,FHSUMMARY=LOCATION */

104:                IF RLG.LOCN(J1)=6 THEN D4='MRES';
105:                IF RLG.LOCN(J1)=7 THEN D4='NMON';
106:                PRINTER='**INVALID CLASSIFICATION,'||RLG.SYSTEM||
107:                  RLG.DUMMY2||',,'||D4;
108:                CALL PRINTX (F(1));
109:                END;
110:          END;
111:       MLGES(RLG.FORHWY_#) = MLGES(RLG.FORHWY_#) + MLGE;
112:       IF RLG.SYSTEM='S' THEN TOTALS(2) = TOTALS(2) + MLGE;
113:          ELSE TOTALS(1) = TOTALS(1) + MLGE;
114:       GOTO READ_RLG_RECORD;

115:       /*** PRINT THE SUMMARY (NON-ZERO LINES ONLY) ****/

116: PRINT_SUMMARY:
117:       DO J1=1 TO 65;
118:          IF MLGES(J1).ROUTE_PRIM=0 & MLGES(J1).ROUTE_SEC=0 THEN GOTO NEXT;
119:          MLGE = MLGES(J1);
120:          OUT_STRUCT.FORHWY_# = J1;
121:          OUT_STRUCT.MILEAGE = MLGE;
122:          OUT_STRUCT.GRAND_TOTAL = MLGE.CONST + MLGE.UNIMP + MLGE.WYE;
123:          PRINTER = STRING(OUT_STRUCT);
124:          CALL PRINTX (F(1));
125: NEXT:
126:          END;

127:       /*** PRINT THE TOTALS ***/
128:       TOTALS(3) = TOTALS(1) + TOTALS(2);
129:       DO J1=1 TO 3;
130:          OUT_STRUCT.MILEAGE = TOTALS(J1);
131:          OUT_STRUCT.GRAND_TOTAL = TOTALS(J1).CONST + TOTALS(J1).UNIMP +
132:             TOTALS(J1).WYE;
133:          PRINTER = STRING(OUT_STRUCT);
134:          SUBSTR(PRINTER,1,8) = CODE(J1);
135:          CALL PRINTX (F(JX(J1)));
136:          END;


137: RETURN:
138:       CLOSE FILE (ROADLOG);
139:       CALL EXIT (PARM);

140: END FORHWY;
```

FHSUMMARY=SURF-TYPE:

Member Name	NERS	
Language	PL/I	
Subroutines	PRINTX1	
	SRTYPR	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- FORHWY-SUMMARY output
	ROADLOG	-- Roadlog file
	SURFTBL	-- Surface type table (SRTYPR)
Instruction	1 - 4	"NERS"

NERS is the second forest highway summarization program, which computes a summary based on forest highway number and surface type. As in NERL, the entire Roadlog file is scanned for forest highway records, and the values accumulated in an array MLGE. Subroutine SRTYPR is used to transform the 4-digit surface type code into a 1-digit classification between 1 and 8. Primary and Interstate mileage is kept separate from Secondary mileage for the summary. After setting up the values in MLGE, the summary is printed.

The NERS program listing follows:

```
/* :FORHWY_SUMMARY,REPORT=ROADLOG,FHSUMMARY=SURF_TYPE */

  1: /* :FORHWY_SUMMARY,REPORT=ROADLOG,FHSUMMARY=SURF_TYPE */
  2: FORHWY:  PROCEDURE (PARM) OPTIONS (MAIN);

  3: /***** DECLARATION OF VARIABLES (ALPHABETICAL ORDER) *****/

  4: DECLARE
  5:    #_HDGS PIC'Z' DEF INSTR POS(72),
  6:    BLANKS CHAR(132) INIT(' '),
  7:    CNTR BIN FIXED,
  8:    F(0:9) PIC'Z' STATIC INIT (0,1,2,3,4,5,6,7,8,9),
  9:    HDG(8,2) CHAR(10),
 10:    HEADING(9) CHAR(132) EXT,
 11:    INSTR CHAR(80) EXT,
 12:    MLGE(62,2,9) PIC'ZZZVZZZ',
 13:    1  OUT_STRUCT,
 14:       3  FORHWY_# PIC'ZZZZBB',
 15:       3  SYSTEM CHAR(10),
 16:       3  MILES(9) PIC'ZZZZZZV.ZZZ',
 17:    PARM CHAR(100),
 18:    PRINTER CHAR(132) EXT,
 19:    1  RLG,
 20:       3  DUMMY1 CHAR(1),
 21:       3  SYSTEM CHAR(1),
 22:       3  DUMMY2 CHAR(14),
 23:       3  SECTN DEC FIXED(5,3),
 24:       3  DUMMY3 CHAR(68),
 25:       3  FORHWY_# DEC FIXED(3,0),
 26:       3  DUMMY4 CHAR(17),
 27:       3  SURF_TYPE DEC FIXED(5,0),
 28:       3 DUMMY5 CHAR(11),
 29:    ROADLOG FILE RECORD KEYED ENV(INDEXED),
 30:    SYSTEM(3) CHAR(10) INIT(' PRIMARY',' SECONDARY',' TOTAL');

 31: /**** PROGRAM INITIALIZATION *****/

 32:    ON ERROR BEGIN;
 33:       PRINTER = '*** ERROR IN FORHWY_SUMMARY ROUTINE';
 34:       CALL PRINTX (F(3));
 35:       GO TO RETURN;
 36:       END;
 37:    CALL INIT (PARM);
 38:    #_HDGS = 7;
 39:    HEADING(1)=SUBSTR(BLANKS,1,30) || 'SUMMARY OF SURFACE TYPES -- ' ||
 40:       'FOREST HIGHWAY SYSTEM';
 41:    HEADING(3) = SUBSTR(BLANKS,1,29) || 'NET CONSTRUCTED LENGTH -- ' ||
 42:       'BY FOREST HIGHWAY ROUTES';
 43:    HEADING(5) = 'FOREST                PRIM-      UNIM-    GRADED &' ||
 44:       '   GRAVEL  BIT SURF     ROAD        PLANT       P. C. ' ||
 45:       '    TOTAL';
 46:    HEADING(6) = 'RT.NO.  SYSTEM       ITIVE     PROVED    DRAINED ' ||
 47:       '           TREATED      MIX          MIX       CONCRETE';

 48:    /*** INITIALIZE SRTYPR SURF TYPE DECODING ROUTINE ***/
 49:    CALL SRTYPRI;
```

```
/* :FORHWY_SUMMARY,REPORT=ROADLOG,FHSUMMARY=SURF_TYPE */

50:     /*** OPEN FILES ***/
51:     OPEN FILE (ROADLOG) INPUT SEQL;
52:     ON ENDFILE (ROADLOG) GOTO PRINT_SUMMARY;

53:     /*** INITIALIZE VARIABLES AND ARRAYS ***/
54:     MLGE = 0;

55:     /*** SCAN ROADLOG FILE FOR PRIM/SEC DATA ***/

56: READ_RLG_RECORD:
57:     READ FILE (ROADLOG) INTO (RLG);
58:     IF RLG.FORHWY_#=0 THEN GO TO READ_RLG_RECORD;
59:     J1 = RLG.FORHWY_#;
60:     IF RLG.SYSTEM='S' THEN J2 = 2;     ELSE J2 = 1;
61:     J3 = RLG.SURF_TYPE;
62:     CALL SRTYPRA (J3);
63:     IF J3=0 THEN DO;
64:        PRINTER='*** ILLEGAL SURFACE TYPE DETECTED,'||RLG.SYSTEM||
65:           SUBSTR(RLG.DUMMY2,1,12)||',' || RLG.SURF_TYPE;
66:        CALL PRINTX (F(1));
67:        GO TO READ_RLG_RECORD;
68:        END;
69:     MLGE(J1,J2,J3) = MLGE(J1,J2,J3) + RLG.SECTN;
70:     MLGE(J1,J2,9) = MLGE(J1,J2,9) + RLG.SECTN;
71:     GO TO READ_RLG_RECORD;

72:     /*** PRINT THE SUMMARY ***/

73: PRINT_SUMMARY:

74:     /*** PRINT THE NON-ZERO LINES OF SUMMARY ***/

75:     DO J1=1 TO 62;
76:        CNTR = 0;
77:        J2 = 2;
78:        OUT_STRUCT.FORHWY_# = J1;
79:        CALL PRINT_PRIMARY (J1);
80:        CALL PRINT_SECONDARY (J1);
81:        IF CNTR>1 THEN CALL PRINT_TOTALS (J1);
82:        END;

83: RETURN:
84:     CLOSE FILE (ROADLOG);
85:     CALL EXIT (PARM);
86:     RETURN;


87: /***** SUBROUTINE TO DO ACTUAL PRINTING *****/


88: PRINT_PRIMARY:  PROCEDURE (J3);
89:     J4 = 1;
90:     GO TO ENTRY;

91: PRINT_SECONDARY:  ENTRY (J3);
92:     J4 = 2;
93: ENTRY:
```

```
/* :FORHWY_SUMMARY,REPORT=ROADLOG,FHSUMMARY=SURF_TYPE */

 94:     OUT_STRUCT.MILES = MLGE(J3,J4,*);
 95:     IF J3¬=62 THEN MLGE(62,J4,*) = MLGE(62,J4,*) + MLGE(J3,J4,*);
 96:     GO TO CONTINUE;

 97: PRINT_TOTALS:  ENTRY (J3);
 98:     J4 = 3;
 99:     OUT_STRUCT.MILES = MLGE(J3,1,*) + MLGE(J3,2,*);


100: CONTINUE:
101:     IF OUT_STRUCT.MILES(9)=0 THEN RETURN;
102:     OUT_STRUCT.SYSTEM = SYSTEM(J4);
103:     PRINTER = STRING(OUT_STRUCT);
104:     IF J3=62 & J4=1 THEN SUBSTR(PRINTER,1,6) = 'TOTALS';
105:     CALL PRINTX (F(J2));
106:     J2 = 1;
107:     CNTR = CNTR + 1;
108:     OUT_STRUCT.FORHWY_# = 0;
109:     END PRINT_PRIMARY;

110: END FORHWY;
```

SUM-LOOPS-&-SPURS --

Member Name	NFR
Language	PL/I
Subroutines	PRINTX1
Files	SYSPRINT -- IBM messages PRINTER -- SUM-LOOPS-&-SPURS output ROADLOG -- Roadlog file SPURTBL -- Table of spurs and loops
Instruction	1 - 3 "NFR"

SUM-LOOPS-&-SPURS prints a summary with one line for each spur or loop on the Primary system. It is not possible to determine directly from the Roadlog file exactly where spurs and loops end; hence, a table is used which specifies the beginning and ending keys of each spur and loop. The table also contains a description of each spur and loop which is printed in the summary. Out-of-state mileage is not included -- each record in the spur/loop is tested for remark codes " ," "SP," "LP," and "NE." The mileage is accumulated in an array M, whose subscript indicates:

1 Route length
2 Construction length
3 Unimproved length
4 Wye length
5 Municipal
6 County
7 National Forest
8 Indian Reservation
9 Other
10 Total rural mileage

After scanning an entire loop or spur, its values are printed, and the next record of SPURTBL read. After the end-of-file is raised on SPURTBL, a line of totals are printed.

The NFR program listing follows:

```
/* :SUM_LOOPS_&_SPURS,REPORT=ROADLOG */

 1: /* :SUM_LOOPS_&_SPURS,REPORT=ROADLOG */
 2: SUMLPSP:  PROCEDURE (PARM) OPTIONS (MAIN);

 3: /* PRINT SUBROUTINE */
 4: DECLARE
 5:    PARM CHAR(100),
 6:    INSTR CHAR(80) EXT,
 7:    #_HDGS PIC'Z' DEF INSTR POS(72),
 8:    BLANKS CHAR(50) INIT(' '),
 9:    HEADING(9) CHAR(132) EXT,
10:    F(0:9) PIC'Z' STATIC INIT (0,1,2,3,4,5,6,7,8,9),
11:    PRINTER CHAR(132) EXT;

12: /* INPUT/OUTPUT VARIABLES */
13: DECLARE
14:    ROADLOG FILE RECORD KEYED ENV(INDEXED),
15:    TABLE FILE RECORD,
16:    1  RLG BASED (PTR_RLG),
17:       3  DUM1 CHAR(1),
18:       3  KEY CHAR(13),
19:       3 REMARK CHAR(2),
20:       3 (SECTN,ROUTE,CONST,UNIMP) DEC FIXED(5,3),
21:       3 WYE DEC FIXED(3,3),
22:       3 DUM2 CHAR(61),
23:       3 LOCN(2) DEC FIXED(3,0),
24:    1  IN BASED (PTR_TBL),
25:       3  DESCR CHAR(20),
26:       3  RT_# CHAR(4),
27:       3  DUM1 CHAR(1),
28:       3  MIPOST1 CHAR(9),
29:       3  DUM2 CHAR(1),
30:       3  MIPOST2 CHAR(9),
31:    1  IN1 BASED (PTR_TBL),
32:       3  DUM1 CHAR(21),
33:       3  RT_# PIC'ZZZ',
34:    1  OUT DEF STRING_OUT,
35:       3  RT_# PIC'ZZZZBBB',
36:       3  DESCR CHAR(22),
37:     2  MILEAGES,
38:       3  ROUTE PIC'ZZZV.ZZZ',
39:       3  CONST PIC'ZZZZZV.ZZZ',
40:       3  UNIMP PIC'ZZZZV.ZZZ',
41:       3  WYE PIC'ZZZV.ZZZ',
42:       3  CITY PIC'ZZZZZZZV.ZZZ',
43:       3  CNTY PIC'ZZZZV.ZZZ',
44:       3  NFOR PIC'ZZZZV.ZZZ',
45:       3  IRES PIC'ZZZZV.ZZZ',
46:       3  OTHER PIC'ZZZV.ZZZ',
47:       3  RURAL PIC'ZZZZZZV.ZZZ',
48:    STRING_OUT CHAR(132) STATIC;

49: /* OTHER VARIABLES */
50: DECLARE
51:    CARRIAGE DEC FIXED (1),
52:    ENDKEY CHAR(16) STATIC,
53:    STARTKEY CHAR(16) STATIC,
54:    M(10) PIC'ZZZVZZZ' STATIC,
55:    1  M_STRCT DEF M,
```

```
/* :SUM_LOOPS_&_SPURS,REPORT=ROADLOG */

 56:       3 (MA,MB,MC,MD,ME,MF,MG,MH,MI,MJ) PIC'ZZZVZZZ',
 57:    TOT(10) PIC'ZZZVZZZ' STATIC;


 58: /***** INITIALIZATION *****/

 59:    ON ERROR BEGIN;
 60:       PRINTER = '*** ERROR IN SUM_LOOPS_&_SPURS ROUTINE';
 61:       CALL PRINTX (F(3));
 62:       GOTO RETURN;
 63:       END;
 64:    CALL INIT (PARM);

 65:    /*** SET UP PAGE HEADINGS *****/
 66:    #_HDGS = 5;
 67:    HEADING(1) = SUBSTR(BLANKS,1,30) ||
 68:       'SUMMARY OF LOOPS AND SPURS ON FEDERAL AID PRIMARY SYSTEM';
 69:    HEADING(3) = 'ROUTE                                                ' ||
 70:       '                                    NATL   INDIAN          ' ||
 71:       '      NET';
 72:    HEADING(4) = ' NO.      LOCATION              ROUTE    CONST' ||
 73:       '   UNIMP    WYE      MUNIC  COUNTY  FOREST  RESERVE  OTHER' ||
 74:       '     RURAL';

 75:    /*** INIT VAR ***/
 76:    CARRIAGE = 1;
 77:    TOT = 0;
 78:    M = 0;
 79:    STRING_OUT = ' ';

 80:    /*** INIT FILES ***/
 81:    OPEN FILE (TABLE) INPUT SEQL RECORD TITLE ('SPURTBL');
 82:    OPEN FILE (ROADLOG) INPUT SEQL;
 83:    ON ENDFILE (TABLE) GOTO FINISH;


 84: /***** MAIN EXECUTION LOOP *****/


 85: LOOP:
 86:    READ FILE (TABLE) SET (PTR_TBL);
 87:    STARTKEY = IN.RT_# || IN.MIPOST1;
 88:    ENDKEY = IN.RT_# || IN.MIPOST2;
 89:    READ FILE (ROADLOG) SET (PTR_RLG) KEY (STARTKEY);
 90:    DO WHILE (RLG.KEY<ENDKEY);
 91:       IF RLG.REMARK='  ' | RLG.REMARK='SP' | RLG.REMARK='LP' |
 92:          RLG.REMARK='NE'
 93:          THEN DO;
 94:          M(1) = M(1) + RLG.ROUTE;
 95:          M(2) = M(2) + RLG.SECTN - RLG.UNIMP - RLG.WYE;
 96:          M(3) = M(3) + RLG.UNIMP;
 97:          M(4) = M(4) + RLG.WYE;
 98:          DO I=1 TO 2;
 99:             IF RLG.LOCN(I)=1 THEN M(5)=M(5)+RLG.SECTN;
100:               ELSE IF RLG.LOCN(I)=2 THEN M(6)=M(6)+RLG.SECTN;
101:               ELSE IF RLG.LOCN(I)=3 THEN M(7)=M(7)+RLG.SECTN;
102:               ELSE IF RLG.LOCN(I)=4 THEN M(8)=M(8)+RLG.SECTN;
103:               ELSE IF RLG.LOCN(I)¬=0 THEN M(9)=M(9)+RLG.SECTN;
```

```
/* :SUM_LOOPS_&_SPURS,REPORT=ROADLOG */

104:              END;
105:                IF RLG.LOCN(1)¬=1 THEN M(10)=M(10)+RLG.SECTN;
106:           END;
107:        READ FILE (ROADLOG) SET (PTR_RLG);
108:        END;
109:     OUT.RT_# = IN1.RT_#;
110:     OUT.DESCR = IN.DESCR;
111:     OUT.MILEAGES = M_STRCT;
112:     PRINTER = STRING_OUT;
113:     CALL PRINTX (F(1));
114:     TOT = TOT + M;
115:     M = 0;
116:     GOTO LOOP;


117: FINISH:
118:     STRING_OUT = 'TOTAL';
119:     M = TOT;
120:     OUT.MILEAGES = M_STRCT;
121:     PRINTER = STRING_OUT;
122:     CALL PRINTX (F(2));


123: RETURN:
124:     CLOSE FILE (TABLE);
125:     CLOSE FILE (ROADLOG);
126:     CALL EXIT (PARM);

127: END SUMLPSP;
```

CHAPTER 2-III
TRAFFIC AND TRUE MILEAGE PROGRAMMER INFORMATION

Introduction

This chapter presents a description of programs comprising the Traffic and True Mileage subsystems of HIS. It is designed for utilization with the publication Highway Information System Volume 1: User Information.

Traffic File Description

Data Set Name	HIS.TRAFFIC
Organization	Indexed Sequential
Logical Record Length	80
Physical Record Length	1280
Key Length	13
Volume Serial Number	231428

The internal format of a Traffic record is shown in Figure 2-III-1. Most of the numeric fields are stored in packed decimal format to conserve storage and improve efficiency. The route number, reference post, and distance from reference post are stored in character format in the key for ease in cross-referencing other HIS files, and also in packed decimal format for computational efficiency.

In addition to the traffic count and descriptor records in the file coded by the user, a "year" record with key "A000" (followed by 9 blanks) exist. This record contains in character format the years for which data in the files refer. The oldest year is in positions 15-16, the next in positions 17-18, the next in positions 19-20, and the latest year in positions 21-22. Positions 23-80 of this record contain blanks.

```
1   TRAFFIC_RECORD,
    2  DELETE_CHARACTER                    CHAR(1),
    2  KEY,
       3  ROUTE_SYSTEM                     CHAR(1),
       3  ROUTE_NUMBER                     CHAR(3),
       3  REFERENCE_POST                   CHAR(3),
       3  DISTANCE                         CHAR(6),
    2  ROUTE_NUMBER                        DEC FIXED (3,0),
    2  REFERENCE_POST                      DEC FIXED (3,0),
    2  DISTANCE                            DEC FIXED (5,3),
    2  ACTUAL_ESTIMATED_CODE               CHAR(1),
    2  REMARK                              CHAR(1),
    2  FIRST_YEAR,
       3  YEAR                             DEC FIXED (2,0),
       3  AVERAGE_DAILY_TRAFFIC            DEC FIXED (5,0),
       3  PERCENT_OUT_OF_STATE             DEC FIXED (3,3),
       3  PERCENT_PICKUPS                  DEC FIXED (3,3),
       3  PERCENT_COMMERCIAL               DEC FIXED (3,3),
    2  SECOND_YEAR                         LIKE FIRST_YEAR,
    2  THIRD_YEAR                          LIKE FIRST_YEAR,
    2  FOURTH_YEAR                         LIKE FIRST_YEAR,
    2  FUTURE_FACTOR                       DEC FIXED (3,3),
    2  DESIGN_HOUR_VOLUME                  DEC FIXED (3,3),
    2  DATE_OF_UPDATE,
       3  YEAR                             CHAR(2),
       3  MONTH                            CHAR(2),
       3  DAY                              CHAR(2),
    2  DUMMY                               CHAR(3);
```

Figure 2-III-1. Traffic file structure.

True Mileage File Description

Data Set Name	HIS.TRUMILE
Organization	Indexed Sequential
Logical Record Length	16
Physical Record Length	1280
Key Length	7
Volume Serial Number	231428

The internal format of a True Mileage record is shown in Figure 2-III-2.

Traffic Summary File Description

Data Set Name	HIS.TRAFSUM
Organization	Indexed Sequential
Logical Record Length	96
Physical Record Length	1632
Key Length	13
Volume Serial Number	231428

The Traffic Summary file is generated by program CREATE-TRAFSUB from the Traffic and True Mileage files. The record format, in PL/I terminology, is shown in Figure 2-III-3.

Four types of records occur in the file: year records, section records, descriptor records, and totals records.

There is one year record in the file. This record contains key "A000" (followed by 9 blanks), and contains in character format the years for which the data in the file refers. The oldest year is in positions 15-16, the next in positions 17-18, and the latest in positions 19-20. The remainder of the year record contains blanks.

One section record appears in the Traffic Summary file for each section defined in the Traffic file. The remark code contains a "W," "T," "O," or "N" for rural, municipal, out-of-state, and non-existent sections, respectively.

A number of descriptor records occur in the Traffic Summary file. These records contain only a key and a remark code. A record containing a "C," "S,"

```
1   TRUE_MILEAGE_RECORD,
    2  DELETE_CHARACTER                  CHAR(1),
    2  KEY,
       3  ROUTE_SYSTEM                   CHAR(1),
       3  ROUTE_NUMBER                   CHAR(3),
       3  REFERENCE_POST                 CHAR(3),
    2  TRUE_MILEAGE                      DEC FIXED (7,3),
    2  DATE_OF_UPDATE                    DEC FIXED (6,0);
```

Figure 2-III-2. True mileage file structure.

```
1   TRAFFIC_SUMMARY_RECORD,
    2  DELETE_CHARACTER                          CHAR(1),
    2  KEY,
       3  ROUTE_SYSTEM                           CHAR(1),
       3  ROUTE_NUMBER                           CHAR(3),
       3  REFERENC_POST                          CHAR(3),
       3  DISTANCE                               CHAR(6),
    2  REMARK                                    CHAR(1),
    2  SECTION_LENGTH                            DEC FIXED (7,3),
    2  FIRST_YEAR,
       3  VEHICLE_MILEAGE                        DEC FIXED (11,3),
       3  OUT_OF_STATE_VEHICLE_MILEAGE           DEC FIXED (11,3),
       3  PICKUPS_VEHICLE_MILEAGE                DEC FIXED (11,3),
       3  COMMERCIAL_VEHICLE_MILEAGE             DEC FIXED (11,3),
    2  SECOND_YEAR                               LIKE FIRST_YEAR,
    2  THIRD_YEAR                                LIKE FIRST_YEAR,
    2  DUMMY                                     CHAR(5);
```

Figure 2-III-3. Traffic summary file structure.

or "L" code occurs for each "C," "S," or "L" record in the Traffic File. A record containing a "D" code occurs just prior to each of the "C," "S," or "L" records; these allow the retrieval of descriptions from the Roadlog file before the coincident sections, spurs, and loops begin. Finally, a record with an "E" code appears at the end of each route, allowing the retrieval of the end-of-file description from the Roadlog file. After the "E" record of each route is a totals record. The totals record has key "snnn999RURAL," where s is an "I," "P," or "S" indicating the route system and nnn is the Federal Aid route number. The record contains the total rural mileage and vehicle miles of the route. In addition, the last route of each route system is followed by a totals record with key "s999RURAL." This record contains the total rural mileage and vehicle miles for the system.

CONVTRF Subroutine

Object Module Name	CONVTRF
Language	PL/I
Files	SYSPRINT
Entry Points	CONVDEC CONVPIC

Several of the programs in the Traffic and True Mileage subsystem utilize the CONVTRF subroutine. This subroutine is stored, in object module format, in cataloged library HIS.OBJECT.

CONVTRF is used to convert a Traffic record to character format from the internal decimal format, and vice versa. The conversion to character format is used when listing the file in "dump" format. The conversion to decimal format is used when updating the file.

The character format of a traffic record is shown in Figure 2-III-4. The decimal format of a record is shown in Figure 2-III-1.

Rather than passing structures, data is passed to CONVTRF in character strings of length 104 and 80. To convert from character to decimal format, code CALL CONVDEC (CHAR,DEC); to convert from decimal to character format, code CALL CONVPIC (CHAR,DEC); CHAR is declared as CHAR(104), and DEC is declared as CHAR(80).

The CONVTRF program listing follows:

```
1   CHAR,
    2  DUMMY1                          CHAR(1),
    2  KEY,
       3  ROUTE_SYSTEM                 CHAR(1),
       3  ROUTE_NUMBER                 CHAR(3),
       3  REFERENCE_POST               CHAR(3),
       3  DISTANCE                     CHAR(6),
    2  ACTUAL_ESTIMATED_CODE           CHAR(1),
    2  DUMMY2                          CHAR(6),
    2  FIRST_YEAR,
       3  YEAR                         CHAR(2),
       3  AVERAGE_DAILY_TRAFFIC        CHAR(5),
       3  PERCENT_OUT_OF_STATE         CHAR(3),
       3  PERCENT_PICKUPS              CHAR(3),
       3  PERCENT_COMMERCIAL           CHAR(3),
    2  SECOND_YEAR                     LIKE FIRST_YEAR,
    2  THIRD_YEAR                      LIKE FIRST_YEAR,
    2  FOURTH_YEAR                     LIKE FIRST_YEAR,
    2  FUTURE_FACTOR                   CHAR(3),
    2  DESIGN_HOUR_VOLUME              CHAR(3),
    2  REMARK                          CHAR(1),
    2  DUMMY3                          CHAR(6),
    2  DATE_OF_UPDATE,
       3  YEAR                         CHAR(2),
       3  MONTH                        CHAR(2),
       3  DAY                          CHAR(2);
```

Figure 2-III-4. Character format of traffic record.

```
/* TRAFFIC CONVERSION ROUTINE */

 1: /* TRAFFIC CONVERSION ROUTINE */

 2: CONVTRF:  PROCEDURE;

 3: /* PICTURE STRUCTURE */
 4: DECLARE
 5:    OLDREC CHAR(104),
 6:    1  S1 BASED (PTR1),
 7:       2  DUM1 CHAR(1),
 8:       2  SYSTEM CHAR(1),
 9:       2 (RT_#,MPOST) PIC'999',
10:       2  FRAC PIC'+9V.999',
11:       2  ACT_EST CHAR(1),
12:       2  DUM CHAR(6),
13:       2  DATA(4),
14:          3  YR PIC'ZZ',
15:          3  ADT PIC'ZZZZZ',
16:          3  X(3) PIC'VZZZ',
17:       2 (FUT,DHV) PIC'VZZZ',
18:       2  REMARK CHAR(1),
19:       2  DUM2 CHAR(6),
20:       2  DATE CHAR(6);

21: /* TRAFFIC RECORD */
22: DECLARE
23:    RECORD CHAR(80),
24:    1  R BASED (PTR_REC),
25:       2 (FIL1,SYSTEM) CHAR(1),
26:       2 (RT_#,MPOST) PIC'999',
27:       2  FRAC PIC'+9V.999',
28:       2 (RT,MP) DEC FIXED (3,0),
29:       2  FR DEC FIXED (5,3),
30:       2 (ACT_EST,REMARK) CHAR(1),
31:       2  DATA(4),
32:          3  YR DEC FIXED (3,0),
33:          3  ADT DEC FIXED (5,0),
34:          3  X(3) DEC FIXED (3,3),
35:       2 (FUT,DHV) DEC FIXED (3,3),
36:       2  DATE CHAR(6);


37: /***** ENTRY TO CONVERT PICTURE STRUCTURE TO DECIMAL *****/

38: CONVDEC:  ENTRY (OLDREC,RECORD);

39:    PTR1 = ADDR(OLDREC);
40:    PTR_REC = ADDR(RECORD);
41:    RECORD = ' ';
42:    R = S1, BY NAME;
43:    R.RT = S1.RT_#;
44:    R.MP = S1.MPOST;
45:    R.FR = S1.FRAC;
46:    RETURN;


47: /***** ENTRY TO CONVERT DECIMAL STRUCTURE TO PICTURE STRUCTURE *****/

48: CONVPIC:  ENTRY (OLDREC,RECORD);
```

```
/* TRAFFIC CONVERSION ROUTINE */

 49:     PTR1 = ADDR(OLDREC);
 50:     PTR_REC = ADDR(RECORD);
 51:     OLDREC = ' ';
 52:     S1 = R, BY NAME;
 53:     RETURN;

 54: END CONVTRF;
```

Program Descriptions

Each program in the Traffic and True Mileage subsystem is stored in load module format in cataloged library HIS.LOADLIB, from which it is retrieved for execution by the HIS supervisor when requested. The member name for each program is given with the program description.

This section of the manual presents a write-up on each program in the Traffic and True Mileage subsystem. An attempt has been made in the source listing itself to document the programs by means of appropriate variable names and comments.

DUMP (Traffic file) --

Member Name	DMT	
Language	PL/I	
Subroutines	PRINTX1	
	CONVTRF	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- DUMP output
	TRAFFIC	-- Traffic file
Instruction	1 - 3	"DMT"
	40 - 52	Beginning key
	56 - 68	Ending key

DUMP provides an unformatted listing of data in the Traffic file, between user-specified (by means of the DATA parameter) records in the file. Subroutine CONVTRF is used to convert the Traffic records into character format for printer. Subroutine PRINTX1 is used for printer output, allowing the use of HIS formatting options in printing.

The DMT program listing follows:

```
/* TRAFFIC FILE DUMP */

 1: /* TRAFFIC FILE DUMP */

 2: DUMP:  PROCEDURE (PARM) OPTIONS (MAIN);

 3: /* INSTRUCTION AND PRINT ROUTINE */
 4: DECLARE
 5:    PARM CHAR(100),
 6:    INSTR CHAR(80) EXT,
 7:    STARTKEY CHAR(13) DEF INSTR POS(40),
 8:    ENDKEY CHAR(13) DEF INSTR POS(56),
 9:    #_HDGS PIC'Z' DEF INSTR POS(72),
10:   (PRINTER,HEADING(9)) CHAR(132) EXT,
11:    PRINTX ENTRY (PIC'Z');

12: /* DATA INPUT */
13: DECLARE
14:    RECORD CHAR(80) BASED (PTR_TRF),
15:    PRINT CHAR(104),
16:    TRAFFIC FILE.RECORD KEYED INPUT SEQL ENV (INDEXED);


17: /***** INITIALIZATION *****/

18:    CALL INIT (PARM);
19:    #_HDGS = 2;
20:    HEADING(1) = '         TRAFFIC DUMP';
21:    OPEN FILE (TRAFFIC);
22:    ON ENDFILE (TRAFFIC) BEGIN;
23:       PRINTER = '   END OF FILE.';
24:       CALL PRINTX (3);
25:       GOTO DONE;
26:       END;
27:    READ FILE (TRAFFIC) SET (PTR_TRF) KEY (STARTKEY);


28: /***** EXECUTION LOOP *****/

29: LOOP:
30:    CALL CONVPIC (PRINT,RECORD);
31:    PRINTER = PRINT;
32:    CALL PRINTX (1);
33:    READ FILE (TRAFFIC) SET (PTR_TRF);
34:    IF SUBSTR(RECORD,2,13)<=ENDKEY THEN GOTO LOOP;

35: DONE:
36:    CLOSE FILE (TRAFFIC);
37:    CALL EXIT (PARM);

38: END DUMP;
```

LIST (Traffic file) --

Member Name	PFT	
Language	PL/I	
Subroutines	PRINTX1	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- LIST output
	TRAFFIC	-- Traffic file
	ROADLOG	-- Roadlog file
	TRUMILE	-- True Mileage file
Instruction	1 - 3	"PFT"
	40 - 52	Beginning key
	56 - 68	Ending key

LIST provides a listing of Traffic data, in a more easily read formatted version than that provided by DUMP. In addition to Traffic data, Roadlog descriptions (at major section breaks) and True Mileage data is also listed. Data Conversions are performed within LIST, rather than by CONVTRF. LIST shows only three years of data (the three complete years). To view the fourth field, DUMP must be utilized.

The PFT program listing follows:

```
/* :LIST,FILE=TRAFFIC,DATA=XXXXX */
```

```
  1: /* :LIST,FILE=TRAFFIC,DATA=XXXXX */

  2: LIST:  PROCEDURE (PARM) OPTIONS (MAIN);


  3: /***** VARIABLE DECLARATIONS (ALPHABETICAL ORDER) *****/

  4: DECLARE
  5:    #_HDGS PIC'Z' DEF INSTR POS(72),
  6:    ENDKEY CHAR(16) DEF INSTR POS(56),
  7:    F(0:9) PIC'Z' STATIC INIT (0,1,2,3,4,5,6,7,8,9),
  8:    1  FTRF STATIC,
  9:       3  KEY CHAR(13),
 10:       3  DUMMY2 CHAR(1) INIT (' '),
 11:       3  TRUE_MILEAGE PIC'ZZZV.ZZZ',
 12:       3  DUMMY3 CHAR(1) INIT (' '),
 13:       3  DESCR CHAR(35),
 14:       3  DUMMY5 CHAR(1) INIT (' '),
 15:       3  DATA(3),
 16:          5  YEAR PIC'ZZ',
 17:          5  ADT PIC'ZZZZZZ',
 18:          5 (OUT,PIC,COM) PIC'ZZZZ',
 19:          5  DUMMY CHAR(1) INIT (' '),
 20:       3  FUT PIC'ZZZ',
 21:       3  DHV PIC'ZZZZ',
 22:       3  DUMMY7 CHAR(1) INIT (' '),
 23:       3  REMARK CHAR(1),
 24:       3  DUMMY8 CHAR(1) INIT (' '),
 25:    HEADING(9) CHAR(132) EXT,
 26:    INSTR CHAR(80) EXT,
 27:    PARM CHAR(100),
 28:    PRINTER CHAR(132) EXT,
 29:    1  RLG BASED (PTR_RLG),
 30:       3  DUM1 CHAR(30),
 31:       3  DESCR CHAR(35),
 32:    ROADLOG FILE RECORD KEYED ENV(INDEXED),
 33:    STARTKEY CHAR(16) DEF INSTR POS(40),
 34:    STRING_FTRF CHAR(130) DEF FTRF,
 35:    TRAFFIC FILE RECORD KEYED ENV(INDEXED),
 36:    1  TRM STATIC,
 37:       3  DUM1 CHAR(8),
 38:       3  TRUE_MILEAGE DEC FIXED (7,3),
 39:       3  DUM2 CHAR(4),
 40:    1  TRF BASED (PTR_TRF),
 41:       3  DUM1 CHAR(1),
 42:       3  KEY CHAR(13),
 43:       3  DUM5 CHAR(4),
 44:       3  FRACTION DEC FIXED (5,3),
 45:       3  ACT_EST CHAR(1),
 46:       3  REMARK CHAR(1),
 47:       3  DATA(3),
 48:          5  YEAR DEC FIXED (3,0),
 49:          5  ADT DEC FIXED (5,0),
 50:          5 (OUT,PIC,COM) DEC FIXED (3,0),
 51:       3  DUM3 CHAR(11),
 52:      3 (FUT,DHV) DEC FIXED (3,0),
 53:    TRUMILE FILE RECORD KEYED ENV(INDEXED);
```

```
/* :LIST,FILE=TRAFFIC,DATA=XXXXX */


54: /***** PROGRAM INITIALIZATION *****/

55:    CALL INIT (PARM);

56:    /*** SET UP COLUMN HEADINGS ***/
57:    #_HDGS = 3;
58:    HEADING(1) = '                    TRUE   ' ||
59:       '                                        **** FIRST YEAR **** ' ||
60:       '*** SECOND YEAR **** **** THIRD YEAR ****';
61:    HEADING(2) = '    KEY          MILEAGE ' ||
62:       '*****  SECTION  DESCRIPTION  ****** YR  ADT  OUT PIC COM ' ||
63:       'YR  ADT  OUT PIC COM YR  ADT  OUT PIC COM FUT DHV';

64:    /*** DD STATEMENTS REQUIRED:  ROADLOG, TRAFFIC, TRUMILE ***/
65:    OPEN
66:       FILE (ROADLOG),
67:       FILE (TRAFFIC),
68:       FILE (TRUMILE);
69:    ON ENDFILE (TRAFFIC) GOTO #RETURN;
70:    READ FILE (TRAFFIC) SET (PTR_TRF) KEY (STARTKEY);
71:    ON KEY (ROADLOG) RLG.DESCR = '**** ROADLOG RECORD MISSING ****';
72:    ON KEY (TRUMILE) TRM.TRUE_MILEAGE = 0;

73: #LOOP:
74:    GOTO PRNT_TRF;

75: READ_TRF:
76:    READ FILE (TRAFFIC) SET (PTR_TRF);
77:    IF TRF.KEY<=ENDKEY THEN GOTO #LOOP;

78: #RETURN:
79:    CLOSE
80:       FILE (ROADLOG),
81:       FILE (TRAFFIC),
82:       FILE (TRUMILE);
83:    CALL EXIT (PARM);
84:    RETURN;


85: /***** SUBROUTINE TO FORMAT TRAFFIC DATA AND PRINT *****/

86: PRNT_TRF:
87:    STRING_FTRF = ' ';

88:    /*** FORMAT RECORD INTO STRUCTURE FTRF FOR PRINTING ***/

89:    FTRF = TRF, BY NAME;
90:    IF TRF.REMARK¬='S' & TRF.REMARK¬='L' & TRF.REMARK¬='C' THEN DO;
91:       READ FILE (TRUMILE) INTO (TRM) KEY (TRF.KEY);
92:       FTRF.TRUE_MILEAGE = TRM.TRUE_MILEAGE + FRACTION;
93:       END;
94:    IF TRF.REMARK='T' | TRF.REMARK='W' |
95:       TRF.REMARK='N' | TRF.REMARK='O' |
96:       TRF.REMARK='S' | TRF.REMARK='L' | TRF.REMARK='C'
97:       THEN DO;
98:       READ FILE (ROADLOG) SET (PTR_RLG) KEY (TRF.KEY);
```

```
/* :LIST,FILE=TRAFFIC,DATA=XXXXX */

 99:        FTRF.DESCR = RLG.DESCR;
100:        END;
101:     PRINTER = STRING_FTRF;
102:     CALL PRINTX (F(1));
103:     GOTO READ_TRF;

104: END LIST;
```

UPDATE (Traffic file) -- UPDATE is comprised of four separate programs, one for each of the functions DELETE, INSERT, REWRITE, and NEW-KEY. The names of the routines are formed by the letters "PDT" followed by the first letter of the function ("D," "I," "R," or "N").

FUNCTION=DELETE:

Member Name	PDTD	
Language	PL/I	
Subroutines	PRINTX1	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- UPDATE output
	TRAFFIC	-- Traffic file
	any name	-- Traffic data cards
Instruction	1 - 4	"PDTD"
	24 - 31	Name of input DD statement

The user codes, by means of the DDNAME parameter on the UPDATE command, the name of the DD statement he will supply with deletion data. Each card read via this DD statement contains, in columns 1-13, the key of a record to be deleted. The program operates with a direct update file; each time a card is read, a PL/I DELETE statement specifying the key on the card is executed. If the key condition is raised (the specified record did not exist in the file), an error message is printed. Each data card is printed as it is read.

The PDTD program listing follows:

```
/* :UPDATE,FILE=TRAFFIC,FUNCTION=DELETE,DDNAME=XXXXX */

 1: /* :UPDATE,FILE=TRAFFIC,FUNCTION=DELETE,DDNAME=XXXXX */

 2: DELETE:  PROCEDURE (PARM) OPTIONS (MAIN);

 3: /* INSTRUCTION */
 4: DECLARE
 5:    INSTR CHAR(80) EXT,
 6:    #_HDGS PIC'Z' DEF INSTR POS(72),
 7:    DDNAME CHAR(8) DEF INSTR POS(24);

 8: /* PRINT ROUTINE */
 9: DECLARE
10:    PARM CHAR(100),
11:   (HEADING(9),PRINTER) CHAR(132) EXT,
12:    PRINTX ENTRY (PIC'Z');

13: /* PERMANENT FILE */
14: DECLARE
15:    PERMDD CHAR(8) STATIC INIT ('TRAFFIC'),
16:    PERM FILE RECORD KEYED ENV (INDEXED);
17: DECLARE
18:    DATA FILE RECORD,
19:    KEY CHAR(80);


20: /***** INITIALIZATION *****/
21:    CALL INIT (PARM);

22:    /* SET UP HEADINGS */
23:    #_HDGS = 2;
24:    HEADING(1) = PERMDD || 'FILE UPDATE -- DELETION OF RECORDS';

25:    /* OPEN FILES */
26:    OPEN FILE (DATA) INPUT RECORD TITLE (DDNAME);
27:    ON ENDFILE (DATA) GOTO CLOSE;
28:    OPEN FILE (PERM) UPDATE DIRECT TITLE (PERMDD);
29:    ON KEY (PERM) BEGIN;
30:       PRINTER = '*** RECORD DOES NOT EXIST IN FILE';
31:       CALL PRINTX (1);
32:       GOTO READ_DATA;
33:       END;


34: /***** MAIN EXECUTION LOOP *****/

35: READ_DATA:
36:    READ FILE (DATA) INTO (KEY);
37:    IF SUBSTR(KEY,8,1)¬='+' THEN SUBSTR(KEY,8,6) = '+0.' ||
38:       SUBSTR(KEY,8,2) || '0';
39:    PRINTER = '      ' || KEY;
40:    CALL PRINTX (2);
41:    DELETE FILE (PERM) KEY (KEY);
42:    GOTO READ_DATA;

43: CLOSE:
44:    CLOSE FILE (PERM);
45:    CLOSE FILE (DATA);
46:    CALL EXIT (PARM);
```

```
/* :UPDATE,FILE=TRAFFIC,FUNCTION=DELETE,DDNAME=XXXXX */

 47: RETURN:

 48: END DELETE;
```

FUNCTION=INSERT:

Member Name	PDTI	
Language	PL/I	
Subroutines	PRINTX1	
	CONVTRF	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- UPDATE output
	TRAFFIC	-- Traffic file
	any name	-- Traffic data cards
Instruction	1 - 4	"PDTI"
	24 - 31	Name of input DD statement

Data cards, when inserting records, contain a complete record. The only fields that are essential in all Traffic records are the key field and the remark field. Both of the Traffic data cards have spaces for the key, the remark, and the design hour volume. Hence, either a first card, a second card, or a first card-second card sequence may be coded for a record when inserting. If both cards are supplied, each must contain the key. Subroutine CONVTRF converts the data cards into the Traffic decimal format.

The PDTI program listing follows:

```
/* :UPDATE,FILE=TRAFFIC,FUNCTION=INSERT,DDNAME=XXXXX */

 1: /* :UPDATE,FILE=TRAFFIC,FUNCTION=INSERT,DDNAME=XXXXX */

 2: INSERT:  PROCEDURE (PARM) OPTIONS (MAIN);

 3: /* INSTRUCTION AND PRINT ROUTINE */
 4: DECLARE
 5:    PARM CHAR(100),
 6:    INSTR CHAR(80) EXT,
 7:    DDNAME CHAR(8) DEF INSTR POS(24),
 8:   (PRINTER,HEADING(9)) CHAR(132) EXT,
 9:    PRINTX ENTRY (PIC'Z');

10: /* DATA INPUT */
11: DECLARE
12:    OLDREC CHAR(104),
13:    C(104) CHAR(1) DEF OLDREC,
14:    CARD CHAR(80),
15:    CA(80) CHAR(1) DEF CARD,
16:    DATA FILE RECORD SEQL INPUT;

17: /* TRAFFIC FILE */
18: DECLARE
19:    RECORD CHAR(80),
20:    TRAFFIC FILE RECORD DIRECT UPDATE KEYED ENV (INDEXED);

21: /* OTHER VARIABLES */
22: DECLARE
23:    FLAG CHAR(1),
24:    KEEP_DATE CHAR(6);


25: /***** INITIALIZATION *****/

26:    CALL INIT (PARM);
27:    #_HDGS = 2;
28:    HEADING(1) = '      TRAFFIC UPDATE -- FUNCTION=INSERT';
29:    OPEN
30:       FILE (DATA) TITLE (DDNAME),
31:       FILE (TRAFFIC);
32:    ON KEY (TRAFFIC) BEGIN;
33:       PRINTER = '*** ATTEMPT TO INSERT OVER EXISTING RECORD';
34:       CALL PRINTX (1);
35:       GOTO READ_DATA;
36:       END;
37:    ON ENDFILE (DATA) FLAG = 'X';
38:    FLAG = 'A';
39:    KEEP_DATE = DATE;


40: /***** EXECUTION LOOP *****/

41: READ_DATA:
42:    IF FLAG='A' THEN READ FILE (DATA) INTO (CARD);
43:    IF FLAG='X' THEN GOTO DONE;
44:    FLAG = 'A';
45:    PRINTER = (5)' ' || CARD;
46:    CALL PRINTX (2);
47:    IF CA(1)='I' | CA(1)='P' | CA(1)='S' THEN GOTO CARD_1;
```

```
/* :UPDATE,FILE=TRAFFIC,FUNCTION=INSERT,DDNAME=XXXXX */

    IF CA(1)='C' THEN GOTO CARD_2;
    PRINTER = '*** UNKNOWN CHARACTER IN COLUMN 1';
    CALL PRINTX (1);
    GOTO READ_DATA;

CARD_1:
    IF CA(8)='+'
       THEN OLDREC = ' ' || SUBSTR(CARD,1,68) || (16)' ' ||
          SUBSTR(CARD,69);
       ELSE OLDREC = ' ' || SUBSTR(CARD,1,7) || '+0.' ||
          SUBSTR(CARD,8,2) || '0' || SUBSTR(CARD,10,7) ||
          SUBSTR(CARD,21,48) || (16)' ' || SUBSTR(CARD,69);
    READ FILE (DATA) INTO (CARD);
    IF CA(1)='C' &
      (CA(9)='+' & SUBSTR(CARD,2,13)=SUBSTR(OLDREC,2,13) |
       CA(9)¬='+' & SUBSTR(CARD,2,9)=SUBSTR(OLDREC,2,9))
       THEN DO;
       PRINTER = (5)' ' || CARD;
       CALL PRINTX (1);
       IF C(9)¬='+' THEN CARD = SUBSTR(CARD,1,8) || '+0.' ||
          SUBSTR(CARD,9,2) || '0' || SUBSTR(CARD,11);
       SUBSTR(OLDREC,76,16) = SUBSTR(CARD,15,16);
       IF SUBSTR(CARD,31,1)¬=' '
          THEN SUBSTR(OLDREC,92,1) = SUBSTR(CARD,31,1);
       IF SUBSTR(CARD,32,3)¬=' '
          THEN SUBSTR(OLDREC,89,3) = SUBSTR(CARD,32,3);
       END;
    ELSE IF FLAG¬='X' THEN FLAG = 'B';
    GOTO TEST_NUMERICS;

CARD_2:
    IF CA(2)¬='I' & CA(2)¬='P' & CA(2)¬='S' THEN DO;
       PRINTER = '*** UNKNOWN CHARACTER IN COLUMN 2';
       CALL PRINTX (1);
       GOTO READ_DATA;
       END;
    IF CA(9)¬='+'
       THEN CARD = SUBSTR(CARD,1,8) || '+0.' || SUBSTR(CARD,9,2) ||
          '0' || SUBSTR(CARD,11);
    OLDREC = ' ' || SUBSTR(CARD,2,13) || (55)' ' || SUBSTR(CARD,15,16)
       || (3)' ' || SUBSTR(CARD,32,3) || SUBSTR(CARD,31,1);

TEST_NUMERICS:
    DO I=3 TO 8,10,12 TO 14,21 TO 91;
       IF (C(I)<'0' | C(I)>'9') & C(I)¬=' ' THEN DO;
          PRINTER = '*** NON-NUMERIC CHARACTER IN NUMERIC FIELD';
          CALL PRINTX (1);
          GOTO READ_DATA;
          END;
       END;
    IF C(92)¬='T' & C(92)¬='W' & C(92)¬='N' & C(92)¬='O' &
       C(92)¬='C' & C(92)¬='S' & C(92)¬='L' & C(92)¬='R' &
       C(92)¬='M' THEN DO;
       PRINTER = '*** INVALID REMARK CODE';
       CALL PRINTX (1);
       GOTO READ_DATA;
       END;
    CALL CONVDEC (OLDREC,RECORD);
```

```
/* :UPDATE,FILE=TRAFFIC,FUNCTION=INSERT,DDNAME=XXXXX */

103:     SUBSTR(RECORD,74) = KEEP_DATE;
104:     WRITE FILE (TRAFFIC) FROM (RECORD) KEYFROM (SUBSTR(RECORD,2));
105:     GOTO READ_DATA;

106: DONE:
107:     PRINTER = '   END OF DATA.';
108:     CALL PRINTX (3);
109:     CLOSE
110:        FILE (TRAFFIC),
111:        FILE (DATA);
112:     CALL EXIT (PARM);

113: END INSERT;
```

FUNCTION=REWRITE:

Member Name	PDTR	
Language	PL/I	
Subroutines	PRINTX1 CONVTRF	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- UPDATE output
	TRAFFIC	-- Traffic file
	any name	-- Traffic data cards
Instruction	1 - 4	"PDTR"
	24 - 31	Name of input DD statement

When rewriting, data cards need contain only the key and the fields which are being altered (FUNCTION=REWRITE cannot alter the key). Either a first card, a second card, or a first card-second card sequence may be supplied for each record rewritten. CONVTRF is used for converting the specified record into character format. The record is then compared to the data cards, and all non-blank fields on the data cards copied into the record (dollar signs first being replaced with blanks). The resultant record is then converted into the Traffic decimal format (by CONVTRF), and a PL/I REWRITE statement used to replace the record.

The PDTR program listing follows:

```
/* :UPDATE,FILE=TRAFFIC,FUNCTION=REWRITE,DDNAME=XXXXX */

 1: /* :UPDATE,FILE=TRAFFIC,FUNCTION=REWRITE,DDNAME=XXXXX */

 2: REWRITE:  PROCEDURE (PARM) OPTIONS (MAIN);

 3: /* INSTRUCTION AND PRINT ROUTINE */
 4: DECLARE
 5:    PARM CHAR(100),
 6:    INSTR CHAR(80) EXT,
 7:    DDNAME CHAR(8) DEF INSTR POS(24),
 8:    #_HDGS PIC'Z' DEF INSTR POS(72),
 9:   (PRINTER,HEADING(9)) CHAR(132) EXT,
10:    PRINTX ENTRY (PIC'Z');

11: /* DATA INPUT */
12: DECLARE
13:    GETREC CHAR(104),
14:    G(104) CHAR(1) DEF GETREC,
15:    OLDREC CHAR(104),
16:    C(104) CHAR(1) DEF OLDREC,
17:    CARD CHAR(80),
18:    CA(80) CHAR(1) DEF CARD,
19:    DATA FILE RECORD SEQL INPUT;

20: /* TRAFFIC FILE */
21: DECLARE
22:    RECORD CHAR(80),
23:    TRAFFIC FILE RECORD DIRECT UPDATE KEYED ENV (INDEXED);

24: /* OTHER VARIABLES */
25: DECLARE
26:    KEEP_DATE CHAR(6);


27: /***** INITIALIZATION *****/

28:    CALL INIT (PARM);
29:    #_HDGS = 2;
30:    HEADING(1) = '      TRAFFIC UPDATE -- FUNCTION=REWRITE';
31:    OPEN
32:       FILE (DATA) TITLE (DDNAME),
33:       FILE (TRAFFIC);
34:    ON KEY (TRAFFIC) BEGIN;
35:       PRINTER = '*** NO RECORD FOR ATTEMPTED REWRITE';
36:       CALL PRINTX (1);
37:       GOTO READ_DATA;
38:       END;
39:    ON ENDFILE (DATA) GOTO DONE;
40:    KEEP_DATE = DATE;


41: /***** EXECUTION LOOP *****/

42: READ_DATA:
43:    READ FILE (DATA) INTO (CARD);
44:    PRINTER = (5)' ' || CARD;
45:    CALL PRINTX (2);
46:    IF CA(1)='I' | CA(1)='P' | CA(1)='S' THEN GOTO CARD_1;
47:    IF CA(1)='C' THEN GOTO CARD_2;
```

```
/* :UPDATE,FILE=TRAFFIC,FUNCTION=REWRITE,DDNAME=XXXXX */

 48:    PRINTER = '*** UNKNOWN CHARACTER IN COLUMN 1';
 49:    CALL PRINTX (1);
 50:    GOTO READ_DATA;

 51: CARD_1:
 52:    IF CA(8)='+'
 53:       THEN OLDREC = ' ' || SUBSTR(CARD,1,68) || (16)' ' ||
 54:          SUBSTR(CARD,69);
 55:       ELSE OLDREC = ' ' || SUBSTR(CARD,1,7) || '+0.' ||
 56:          SUBSTR(CARD,8,2) || '0' || SUBSTR(CARD,10,7) ||
 57:          SUBSTR(CARD,21,48) || (16)' ' || SUBSTR(CARD,69);
 58:    GOTO GET_RECORD;

 59: CARD_2:
 60:    IF CA(2)¬='I' & CA(2)¬='P' & CA(2)¬='S' THEN DO;
 61:       PRINTER = '*** UNKNOWN CHARACTER IN COLUMN 2';
 62:       CALL PRINTX (1);
 63:       GOTO READ_DATA;
 64:       END;
 65:    IF CA(9)¬='+'
 66:       THEN CARD = SUBSTR(CARD,1,8) || '+0.' || SUBSTR(CARD,9,2) ||
 67:          '0' || SUBSTR(CARD,11);
 68:    OLDREC = ' ' || SUBSTR(CARD,2,13) || (55)' ' || SUBSTR(CARD,15,16)
 69:       || (3)' ' || SUBSTR(CARD,32,3) || SUBSTR(CARD,31,1);

 70: GET_RECORD:
 71:    READ FILE (TRAFFIC) INTO (RECORD) KEY (SUBSTR(OLDREC,2));
 72:    CALL CONVPIC (GETREC,RECORD);
 73:    DO I=15 TO 104;
 74:       IF C(I)¬=' ' THEN DO;
 75:          IF C(I)='$' THEN C(I) = ' ';
 76:          G(I) = C(I);
 77:          END;
 78:       END;

 79:    /* TEST NUMERIC FIELDS */
 80:    DO I=3 TO 8,10,12 TO 14,21 TO 91;
 81:       IF (G(I)<'0' | G(I)>'9') & G(I)¬=' ' THEN DO;
 82:          PRINTER = '*** NON-NUMERIC CHARACTER IN NUMERIC FIELD';
 83:          CALL PRINTX (1);
 84:          GOTO READ_DATA;
 85:          END;
 86:       END;
 87:    IF G(92)¬='T' & G(92)¬='W' & G(92)¬='N' & G(92)¬='O' &
 88:       G(92)¬='C' & G(92)¬='S' & G(92)¬='L' & G(92)¬='R' &
 89:       G(92)¬='M' THEN DO;
 90:       PRINTER = '*** INVALID REMARK CODE';
 91:       CALL PRINTX (1);
 92:       GOTO READ_DATA;
 93:       END;
 94:    CALL CONVDEC (GETREC,RECORD);
 95:    SUBSTR(RECORD,74) = KEEP_DATE;
 96:    REWRITE FILE (TRAFFIC) FROM (RECORD) KEY (SUBSTR(RECORD,2));
 97:    GOTO READ_DATA;

 98: DONE:
 99:    PRINTER = '   END OF DATA.';
100:    CALL PRINTX (3);
```

```
/* :UPDATE,FILE=TRAFFIC,FUNCTION=REWRITE,DDNAME=XXXXX */

101:    CLOSE
102:       FILE (TRAFFIC),
103:       FILE (DATA);
104:    CALL EXIT (PARM);

105: END REWRITE;
```

FUNCTION=NEW-KEY:

Member Name	PDTN	
Language	PL/I	
Subroutines	PRINTX1	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- UPDATE output
	TRAFFIC	-- Traffic file
	any name	-- Traffic data cards
Instruction	1 - 4	"PDTN"
	24 - 31	Name of input DD statement

The NEW-KEY function allows alteration of the key field, which cannot be altered by REWRITE. Upon reading a data card (containing the existing key in columns 1-13, and the new key in columns 15-27), the program first checks to be sure that no record already exists with the new key, then reads the existing record, supplies the new key, inserts the record, and deletes the old record. An error message is generated either if a record already exists with the new key, or if no record exists with the old key.

The PDTN program listing follows:

```
/* :UPDATE,FILE=TRAFFIC,FUNCTION_NEW-KEY,DATA=XXXXX */

 1: /* :UPDATE,FILE=TRAFFIC,FUNCTION_NEW-KEY,DATA=XXXXX */

 2: NEWKEY:  PROCEDURE (PARM) OPTIONS (MAIN);

 3: /* INSTRUCTION */
 4: DECLARE
 5:    INSTR CHAR(80) EXT,
 6:    DDNAME CHAR(8) DEF INSTR POS(24),
 7:    #_HDGS PIC'Z' DEF INSTR POS(72);

 8: /* PRINT ROUTINE */
 9: DECLARE
10:    PARM CHAR(100),
11:   (HEADING(9),PRINTER) CHAR(132) EXT,
12:    PRINTX ENTRY (PIC'Z'),
13:    PRINTXA ENTRY (PIC'Z',PIC'ZZ');

14: /* PERMANENT FILE */
15: DECLARE
16:    RECORD CHAR(80) STATIC,
17:    1  R DEF RECORD,
18:       3  DUM1 CHAR(1),
19:       3  KEY CHAR(13),
20:    PERMDD CHAR(8) STATIC INIT ('TRAFFIC'),
21:    PERM FILE RECORD KEYED ENV (INDEXED);

22: /* DATA INPUT */
23: DECLARE
24:    CARD CHAR(80) BASED (PTR_DATA),
25:    1  C BASED (PTR_DATA),
26:       3  OLD CHAR(13),
27:       3  DUM CHAR(1),
28:       3  NEW CHAR(13),
29:    DATA FILE RECORD;


30: /***** INITIALIZATION *****/

31:    CALL INIT (PARM);
32:    #_HDGS = 2;
33:    HEADING(1) = PERMDD || 'FILE UPDATE -- NEW KEY';

34:    /* INIT FILES */
35:    ON UNDEFINEDFILE (DATA) BEGIN;
36:       PRINTER = '*** ' || DDNAME || ' DD STATEMENT MISSING';
37:       CALL PRINTX (3);
38:       GOTO RETURN;
39:       END;
40:    OPEN FILE (DATA) INPUT RECORD TITLE (DDNAME);
41:    OPEN FILE (PERM) UPDATE DIRECT TITLE (PERMDD);
42:    ON ENDFILE (DATA) GOTO FINISH;


43: /***** MAIN EXECUTION LOOP *****/

44: READ_DATA:
45:    READ FILE (DATA) SET (PTR_DATA);
46:    PRINTER = '     ' || CARD;
```

```
/* :UPDATE,FILE=TRAFFIC,FUNCTION_NEW-KEY,DATA=XXXXX */

47:     CALL PRINTXA (3,7);
48:     ON KEY (PERM) GOTO GET_RECORD;
49:     READ FILE (PERM) INTO (RECORD) KEY (C.NEW);
50:     PRINTER ='*** ATTEMPT TO INSERT OVER EXISTING RECORD';
51:     CALL PRINTX (1);
52:     GOTO READ_DATA;

53: GET_RECORD:
54:     ON KEY (PERM) BEGIN;
55:        PRINTER = '*** RECORD DOES NOT EXIST IN FILE';
56:        CALL PRINTX (1);
57:        GOTO READ_DATA;
58:        END;
59:     READ FILE (PERM) INTO (RECORD) KEY (C.OLD);
60:     R.KEY = C.NEW;
61:     WRITE FILE (PERM) FROM (RECORD) KEYFROM (R.KEY);
62:     DELETE FILE (PERM) KEY (C.OLD);
63:     GOTO READ_DATA;

64: FINISH:
65:     PRINTER = '  END OF DATA';
66:     CALL PRINTX (3);
67:     CLOSE FILE (PERM);
68:     CLOSE FILE (DATA);

69: RETURN:
70:     CALL EXIT (PARM);

71: END NEWKEY;
```

UPDATE-BY-YEAR (Traffic file) --

Member Name	PGT
Language	PL/I
Subroutines	none
Files	SYSPRINT -- IBM and UPDATE-BY-YEAR output TRAFFIC -- Traffic file
Instruction	1 - 3 "PGT"

UPDATE-BY-YEAR goes through the Traffic file, and shifts the data one year; it must be executed yearly after the fourth data year has been filled in. The first record of the Traffic file (key "A000") contains the years for the file; this record is also updated by adding one to each year.

The PGT program listing follows:

```
/* :UPDATE-BY-YEAR,FILE=TRAFFIC */

 1: /* :UPDATE-BY-YEAR,FILE=TRAFFIC */

 2: UPDYR:  PROCEDURE OPTIONS (MAIN);

 3: DECLARE
 4:    TRAFFIC FILE INT RECORD KEYED UPDATE ENV (INDEXED),
 5:    YR(4) PIC'ZZ' DEF TRF POS(15),
 6:    TXPOS CHAR(1) DEF TRF POS(57),
 7:    1  TX BASED (PTR),
 8:       2  YEAR DEC FIXED (3,0),
 9:       2  ADT DEC FIXED (5,0),
10:       2 (A,B,C) DEC FIXED (3,0),
11:    TRF CHAR(80) STATIC,
12:    DTE CHAR(6);

13: /* INIT VAR */
14:    DTE = DATE;
15:    OPEN FILE (TRAFFIC);
16:    PTR = ADDR(TXPOS);
17:    ON ENDFILE (TRAFFIC) GOTO STOP;

18: /* FIRST RECORD CONTAINS DATA YEARS */
19:    READ FILE (TRAFFIC) INTO (TRF);
20:    DO I=1 TO 4;
21:       YR(I) = YR(I) + 1;
22:       END;
23:    REWRITE FILE (TRAFFIC) FROM (TRF);

24: /* EXECUTION LOOP */

25: LOOP:
26:    READ FILE (TRAFFIC) INTO (TRF);
27:    SUBSTR(TRF,24,33) = SUBSTR(TRF,35,33);
28:    TX = 0;
29:    SUBSTR(TRF,72,6) = DTE;
30:    REWRITE FILE (TRAFFIC) FROM (TRF);
31:    GOTO LOOP;

32: STOP:
33:    CLOSE FILE (TRAFFIC);
34:    PUT FILE (SYSPRINT) SKIP EDIT
35:       ('UPDATE-BY-YEAR SUCCESSFULLY COMPLETED') (A);

36: END UPDYR;
```

COPY (Traffic file) --

Member Name	PBT	
Language	PL/I	
Subroutines	PRINTX1	
	CONVTRF	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- COPY output
	TRAFFIC	-- Traffic file
	SAVETRF	-- Backup copy (output)
Instruction	1 - 3	"PBT"
	5	"Y"/"N" for LIST=YES/LIST=NO

COPY prepares a backup copy of the Traffic file. The backup copy is a sequential version of the Traffic file, with identical record length (80 characters). A dummy record containing the date is first written. This record is followed by each of the Traffic records. If LIST=YES is specified, CONVTRF is used to convert the records into character format, and the records listed in "dump" format (identical to that produced by DUMP). A count is taken of the number of records in the file. The count is printed after the last record is written.

The PBT program listing follows:

```
/* :COPY,FILE=TRAFFIC,LIST=XXXXX */

 1: /* :COPY,FILE=TRAFFIC,LIST=XXXXX */

 2: COPY:  PROCEDURE (PARM) OPTIONS (MAIN);

 3: /* INSTRUCTION */
 4: DECLARE
 5:    INSTR CHAR(80) EXT,
 6:    LIST CHAR(1) DEF INSTR POS(5),
 7:    #_HDGS PIC'Z' DEF INSTR POS(72);

 8: /* PRINT ROUTINE */
 9: DECLARE
10:    PARM CHAR(100),
11:   (HEADING(9),PRINTER) CHAR(132) EXT,
12:    PRINTX ENTRY (PIC'Z');

13: /* FILES */
14: DECLARE
15:    RECORD CHAR(80) BASED (PTR),
16:    R CHAR(104),
17:    BACKDD CHAR(8) STATIC INIT ('SAVETRF'),
18:    PERMDD CHAR(8) STATIC INIT ('TRAFFIC'),
19:    PERM FILE RECORD KEYED ENV (INDEXED),
20:    BACKUP FILE RECORD;

21: /* OTHER VARIABLES */
22: DECLARE
23:    UD CHAR(6),
24:    CNTR BIN FIXED (31),
25:    PCNTR PIC'ZZZZZ9';


26: /***** INITIALIZATION *****/

27:    CALL INIT (PARM);

28:    /* SET UP HEADINGS */
29:    #_HDGS = 2;
30:    HEADING(1) = PERMDD || 'FILE COPY ROUTINE';

31:    /* INIT FILES */
32:    OPEN FILE (PERM) INPUT TITLE (PERMDD);
33:    OPEN FILE (BACKUP) OUTPUT TITLE (BACKDD);
34:    ON ENDFILE (PERM) GOTO DONE;

35:    /* RECORD DATE */
36:    UD = DATE;
37:    PTR = ADDR(R);
38:    RECORD = SUBSTR(UD,3,2) || '/' ||
39:         SUBSTR(UD,5,2) || '/' || SUBSTR(UD,1,2);
40:    WRITE FILE (BACKUP) FROM (RECORD);


41: /***** MAIN EXECUTION LOOP *****/

42:    DO CNTR=1 TO 999999;
43:       READ FILE (PERM) SET (PTR);
44:       WRITE FILE (BACKUP) FROM (RECORD);
```

```
/* :COPY,FILE=TRAFFIC,LIST=XXXXX */

45:       IF LIST='Y' THEN DO;
46:       CALL CONVPIC (RECORD,R);
47:       PRINTER = R;
48:          CALL PRINTX (1);
49:          END;
50:       END;

51: DONE:
52:    PCNTR = CNTR - 1;
53:    PRINTER = 'NUMBER OF RECORDS IN FILE:  ' || PCNTR;
54:    CALL PRINTX (3);
55:    CLOSE FILE (PERM);
56:    CLOSE FILE (BACKUP);
57:    CALL EXIT (PARM);

58: END COPY;
```

CREATE (Traffic file) --

Member Name	PAT	
Language	PL/I	
Subroutines	PRINTX1	
	CONVTRF	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- CREATE output
	TRAFFIC	-- Traffic file (output)
	SAVETRF	-- Backup copy
Instruction	1 - 3	"PAT"
	5	"Y"/"N" for LIST=YES/LIST=NO

CREATE restores the Traffic file from a backup copy saved via program COPY. The first record in the file is a dummy record, containing the date on which the file was copied. This date is printed prior to performing the create operation. After printing the date, the records are read from the backup copy and written into the Traffic file, destroying the previous file. If LIST=YES is specified, subroutine CONVTRF is used to convert the records to character format for printing. As with COPY, the records are counted as they are written. The count is printed after the create operation is complete.

The PAT program listing follows:

```
/* :CREATE,FILE=TRAFFIC,LIST=XXXXX */

 1: /* :CREATE,FILE=TRAFFIC,LIST=XXXXX */

 2: CREATE:  PROCEDURE (PARM) OPTIONS (MAIN);

 3: /* INSTRUCTION */
 4: DECLARE
 5:    INSTR CHAR(80) EXT,
 6:    LIST CHAR(1) DEF INSTR POS(5),
 7:    #_HDGS PIC'Z' DEF INSTR POS(72);

 8: /* PRINT ROUTINE */
 9: DECLARE
10:    PARM CHAR(100),
11:   (HEADING(9),PRINTER) CHAR(132) EXT,
12:    PRINTX ENTRY (PIC'Z');

13: /* FILES */
14: DECLARE
15:    RECORD CHAR(104) BASED (PTR),
16:    BACKDD CHAR(8) STATIC INIT ('SAVETRF'),
17:    PERMDD CHAR(8) STATIC INIT ('TRAFFIC'),
18:    PERM FILE RECORD KEYED ENV (INDEXED),
19:    BACKUP FILE RECORD;

20: /* OTHER VARIABLES */
21: DECLARE
22:    CNTR BIN FIXED (31),
23:    PCNTR PIC'ZZZZZ9';


24: /***** INITIALIZATION *****/

25:    CALL INIT (PARM);

26:    /* SET UP HEADINGS */
27:    #_HDGS = 2;
28:    HEADING(1) = PERMDD || 'FILE CREATION ROUTINE';

29:    /* INIT FILES */
30:    OPEN FILE (BACKUP) INPUT TITLE (BACKDD);
31:    OPEN FILE (PERM) OUTPUT TITLE (PERMDD);
32:    ON ENDFILE (BACKUP) GOTO DONE;

33:    /* PRINT DATE */
34:    READ FILE (BACKUP) SET (PTR);
35:    PRINTER = '   DATE OF BACKUP FILE IS ' || RECORD;
36:    CALL PRINTX (1);
37:    PRINTER = ' ';
38:    CALL PRINTX (1);


39: /***** MAIN EXECUTION LOOP *****/

40:    DO CNTR=1 TO 999999;
41:       READ FILE (BACKUP) SET (PTR);
42:       WRITE FILE (PERM) FROM (RECORD) KEYFROM (SUBSTR(RECORD,2));
43:       IF LIST='Y' THEN DO;
44:          PRINTER = '     ' || RECORD;
```

```
/* :CREATE,FILE=TRAFFIC,LIST=XXXXX */

45:           CALL PRINTX (1);
46:           END;
47:        END;

48: DONE:
49:     PCNTR = CNTR - 1;
50:     PRINTER = 'NUMBER OF RECORDS IN FILE:  ' || PCNTR;
51:     CALL PRINTX (3);
52:     CLOSE FILE (PERM);
53:     CLOSE FILE (BACKUP);
54:     CALL EXIT (PARM);

55: END CREATE;
```

LIST (True Mileage file) --

Member Name	PFM	
Language	PL/I	
Subroutines	PRINTX1	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- LIST output
	TRUMILE	-- True Mileage file
Instruction	1 - 3	"PFM"
	40 - 46	Beginning key
	56 - 62	Ending key

LIST provides a formatted listing of True Mileage data in a data range specified by the user (by means of the DATA parameter on the command). Subroutine PRINTX1 is used for the listing, allowing the use of all HIS formatting options in the listing.

The PFM program listing follows:

```
/* :LIST,FILE=TRUMILE,DATA=XXXXX */

  1: /* :LIST,FILE=TRUMILE,DATA=XXXXX */

  2: LIST:  PROCEDURE (PARM) OPTIONS (MAIN);

  3: /* INSTRUCTION AND PRINT ROUTINE */
  4: DECLARE
  5:    PARM CHAR(100),
  6:    INSTR CHAR(80) EXT,
  7:    STARTKEY CHAR(7) DEF INSTR POS(40),
  8:    ENDKEY CHAR(7) DEF INSTR POS(56),
  9:    #_HDGS PIC'Z' DEF INSTR POS(72),
 10:    HEADING(9) CHAR(132) EXT,
 11:    PRINTER CHAR(132) EXT,
 12:    PRINTX ENTRY (PIC'Z');

 13: /* TRUE MILEAGE FILE */
 14: DECLARE
 15:    1  TRM STATIC,
 16:       2  DUM1 CHAR(1),
 17:       2  RT_# CHAR(4),
 18:       2  MPOST CHAR(3),
 19:       2  TRUE DEC FIXED (7,3),
 20:       2  DATE DEC FIXED (6,0),
 21:    TRUMILE FILE INTERNAL RECORD KEYED ENV (INDEXED);

 22: /* OUTPUT VARIABLES */
 23: DECLARE
 24:    C1 PIC'999999',
 25:    YR CHAR(2) DEF C1,
 26:    MONTH CHAR(2) DEF C1 POS(3),
 27:    DAY CHAR(2) DEF C1 POS(5),
 28:    OUT CHAR(132),
 29:    1  O DEF OUT,
 30:       2  RT_# CHAR(7),
 31:       2  MPOST CHAR(8),
 32:       2  TRUE PIC'ZZZV.999BB',
 33:       2  DATE CHAR(8);


 34: /***** INITIALIZATION *****/

 35:    CALL INIT (PARM);
 36:    #_HDGS = 3;
 37:    HEADING(1) = '                    TRUE';
 38:    HEADING(2) = 'RT_#  MPOST    MILEAGE    DATE';
 39:    OPEN FILE (TRUMILE);
 40:    ON KEY (TRUMILE) BEGIN;
 41:       PRINTER = '*** NO TRUMILE RECORD FOR KEY ' || STARTKEY;
 42:       CALL PRINTX (2);
 43:       GOTO STOP;
 44:       END;
 45:    READ FILE (TRUMILE) INTO (TRM) KEY (STARTKEY);
 46:    ON ENDFILE (TRUMILE) BEGIN;
 47:       PRINTER = '  END OF FILE.';
 48:       CALL PRINTX (3);
 49:       GOTO STOP;
 50:       END;
```

```
/* :LIST,FILE=TRUMILE,DATA=XXXXX */


 51: /*** EXECUTION LOOP ***/

 52: LOOP:
 53:    OUT = ' ';
 54:    O.RT_# = TRM.RT_#;
 55:    O.MPOST = TRM.MPOST;
 56:    O.TRUE = TRM.TRUE;
 57:    C1 = TRM.DATE;
 58:    IF TRM.DATE¬=0 THEN O.DATE = MONTH || '/' || DAY || '/' || YR;
 59:    PRINTER = OUT;
 60:    CALL PRINTX (1);
 61:    READ FILE (TRUMILE) INTO (TRM);
 62:    IF TRM.RT_#||TRM.MPOST<=ENDKEY THEN GOTO LOOP;

 63: STOP:
 64:    CLOSE FILE (TRUMILE);
 65:    CALL EXIT (PARM);

 66: END LIST;
```

UPDATE (True Mileage file) -- UPDATE consists of three separate programs: PDMD for deletion, PDMI for insertion, and PDMR for revision of records.

FUNCTION=DELETE:

Member Name	PDMD	
Language	PL/I	
Subroutines	PRINTX1	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- UPDATE output
	TRUMILE	-- True Mileage file
	any name	-- Data cards
Instruction	1 - 4	"PDMD"
	24 - 31	Name of input DD statement

PDMD reads data cards containing a key in columns 1-7. The record corresponding to the specified key is deleted. The data cards are printed as they are read.

The PDMD program listing follows:

```
/**:UPDATE, FILE=TRUMILE, FUNCTION=DELETE DDNAME=XXXXX **/

 1:  /**:UPDATE, FILE=TRUMILE, FUNCTION=DELETE DDNAME=XXXXX **/
 2: CANCEL: PROCEDURE (PARM) OPTIONS (MAIN);
 3: DECLARE
 4:     INSTR CHAR (80) EXT,
 5:     #_HDGS PIC'Z' DEF INSTR POS (72),
 6:     DDNAME CHAR(8) DEF INSTR POS(24),
 7:     PARM CHAR (100),
 8:     (HEADING(9), PRINTER) CHAR(132) EXT,
 9:     PRINTX ENTRY (PIC'Z'),
10:     CNSTDD CHAR(8) STATIC INIT('TRUMILE'),
11:     CNST FILE RECORD KEYED ENV(INDEXED),
12:     NEWDT FILE RECORD,
13:    KEY1 CHAR(16) BASED (PTR_NEWDT);
14: /**INIT**/
15:     CALL INIT (PARM);
16: /** HDGS **/
17:     #_HDGS = 2;
18:     HEADING(1)=CNSTDD||'FILE UPDATE -- DELETION OF RECORDS';
19: /**OPEN FILES**/
20:     OPEN FILE (NEWDT) INPUT RECORD TITLE (DDNAME);
21:     ON ENDFILE(NEWDT) GO TO CLOSE;
22:     OPEN FILE(CNST) UPDATE DIRECT TITLE(CNSTDD);
23:     ON KEY (CNST) BEGIN;
24:       PRINTER = '****RECORD DOES NOT EXIST IN FILE';
25:       CALL PRINTX (1);
26:       GOTO READ_NEWDT;
27:       END;
28: /**MAIN LOOP**/
29:     READ_NEWDT:
30:       READ FILE (NEWDT) SET(PTR_NEWDT);
31:    PRINTER = '      ' || KEY1;
32:     CALL PRINTX (2);
33:    DELETE FILE (CNST) KEY (KEY1);
34:     GO TO READ_NEWDT;
35:     CLOSE:
36:     CLOSE FILE(CNST);
37:     CLOSE FILE(NEWDT);
38:     CALL EXIT (PARM);
39:  END CANCEL;
```

FUNCTION=INSERT:

Member Name	PDMI
Language	PL/I
Subroutines	PRINTX1
Files	SYSPRINT -- IBM messages PRINTER -- UPDATE output TRUMILE -- True Mileage file any name -- Data cards
Instruction	1 - 4 "PDMI" 24 - 31 Name of input DD statement

PDMI reads data cards containing a key in columns 1-7, and a true mileage in columns 8-13. The true mileage is converted to decimal, and the date appended. The record is then inserted into the file.

The PDMI program listing follows:

```
/*** MEMBER NAME PDMI */

  1: /*** MEMBER NAME PDMI */
  2: /**:UPDATE,FILE=TRUMILE,FUNCTION=WRITE, DDNAME=XXXXX **/
  3: INSERT: PROC(PARM) OPTIONS(MAIN);
  4: DECLARE
  5:  #_HDGS PIC'Z' DEF INSTR POS(72),
  6:   INSTR CHAR(80) EXT,
  7:   DDNAME CHAR(8) DEF INSTR POS(24),
  8:   PARM CHAR(100),
  9:   (HEADING (9),PRINTER) CHAR(132) EXT,
 10:   PRINTX ENTRY(PIC'Z'),
 11:   1 CARD BASED(PTR),
 12:    2 KEY CHAR(7), 2 TRUE PIC'ZZZVZZZ',
 13:   1 T1 STATIC,
 14:    2 DUM1 CHAR(1) INIT(' '), 2 KEY CHAR(7),
 15:    2 TRUE DEC FIXED(7,3), 2 DTE DEC FIXED(7,0),
 16:   DATER CHAR(6), DATES PIC'Z7ZZZZ' DEF DATER,
 17:   CNSTDD CHAR(8) STATIC INIT('TRUMILE'),
 18:   CNST FILE RECORD KEYED ENV(INDEXED),
 19:   CD FILE RECORD;
 20: /** PGM INIT **/
 21:   CALL INIT(PARM);
 22:   DATER=DATE;
 23:   T1.DTE=DATES;
 24:   #_HDGS=2;
 25:   HEADING(1)='FILE UPDATE TO TRUMILE FOR NEW RECORDS';
 26: /** OPEN FILES **/
 27:   OPEN FILE(CD) INPUT RECORD TITLE(DDNAME);
 28:   ON ENDFILE(CD) GOTO CLOSE;
 29:   OPEN FILE(CNST) UPDATE DIRECT TITLE(CNSTDD);
 30:   ON KEY(CNST) BEGIN;
 31:   PRINTER='***EXISTING RECORD, ATTEMPT TO INSERT NEW ***';
 32:   CALL PRINTX(1);
 33:   GOTO BRING_IN_DATA;
 34:   END;
 35: /** LOOP **/
 36: BRING_IN_DATA:
 37:   READ FILE(CD) SET(PTR);
 38:   PRINTER='    '||CARD.KEY;
 39:   CALL PRINTX(2);
 40:   T1=CARD, BY NAME;
 41:   WRITE FILE(CNST) FROM(T1) KEYFROM(T1.KEY);
 42:   GOTO BRING_IN_DATA;
 43:   CLOSE:
 44:   CLOSE FILE(CNST);
 45:   CLOSE FILE(CD);
 46:   CALL EXIT(PARM);
 47:   END INSERT;
```

FUNCTION=REWRITE:

Member Name	PDMR	
Language	PL/I	
Subroutines	PRINTX1	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- UPDATE output
	TRUMILE	-- True Mileage file
	any name	-- Data cards
Instruction	1 - 4	"PDMR"
	24 - 31	Name of input DD statement

PDMR reads data cards (same format as for PDMI), and forms a True Mileage record by converting the true mileage to decimal and appending the date. The resultant record is then entered into the file with a REWRITE statement. An optional data card format may be used for adding a constant value (positive or negative) to every record in a range of the file. The data cards have the format:

±nn.nnn,srrr,mmm or

±nn.nnn,srrr,mmm,mmm

±nn.nnn is the constant to be added. Leading zeroes must be supplied if the value is less than 10, and the decimal point is punched. The sign (+ or -) must be in column 1. srrr is the route system ("I," "P," or "S") followed by the route number. Leading zeroes must be coded in the route number to fill three digits. mmm is a reference post. If the first format is used, the program begins with this reference post and continues to the end of the route. If the second format is used (note that the second reference post must be at least as large as the first), the program begins at the first reference post and continues through the second.

The PDMR program listing follows:

```
/**MEMBER NAME PDMR**/

 1:  /**MEMBER NAME PDMR**/
 2:  /**:UPDATE, FILE=TRUMILE,FUNCTION=REWRITE, DDNAME=XXXXX**/
 3: REVISE:  PROCEDURE (PARM) OPTIONS (MAIN);
 4: DECLARE
 5:     INSTR CHAR(80)EXT, PARM CHAR(100),
 6:     #_HDGS PIC'Z' DEF INSTR POS(72),
 7:     DDNAME CHAR(8) DEF INSTR POS(24),
 8:     (HEADING(9),PRINTER) CHAR(132) EXT,
 9:     PRINTX ENTRY(PIC'Z'),
10:     CNSTDD CHAR(8) STATIC INIT('TRUMILE'),
11:     CNST FILE RECORD KEYED ENV(INDEXED),
12:     CD FILE RECORD,
13:     NUMBER PIC'SZZV.ZZZ' BASED(PTR),
14:     C CHAR(30) BASED(PTR),
15:     1 CARD BASED(PTR),
16:      2 KEY CHAR(7), 2 TRUE PIC'ZZZVZZZ',
17:     1 T1 STATIC,
18:      2 DUM1 CHAR(1) INIT(' '), 2 KEY CHAR(7),
19:      2 TRUE DEC FIXED(7,3), 2 DTE DEC FIXED(7,0),
20:     DATER CHAR(6), DATES PIC'ZZZZZZ' DEF DATER;
21:     CALL INIT (PARM);
22:     DATER = DATE;
23:     #_HDGS=2;
24:     HEADING(1)= 'HIS TRM FILE REVISION';
25: /**FILES OPEN**/
26:     OPEN FILE(CD) INPUT RECORD TITLE (DDNAME);
27:     ON ENDFILE(CD) GOTO FINISH;
28:    OPEN FILE (CNST) UPDATE TITLE (CNSTDD);
29:     ON KEY (CNST) BEGIN;
30:       PRINTER = '***RECORD DOES NOT EXIST IN FILE**';
31:     CALL PRINTX (1);
32:    GOTO LOOP;
33:       END;
34: /**MAIN LOOP**/
35:     LOOP:
36:     READ FILE(CD) SET (PTR);
37:    PRINTER = (5)' ' || C;
38:    CALL PRINTX (2);
39:   /* TEST FOR MILEAGE ALTERING RECORD  */
40:     IF SUBSTR(CARD.KEY,1,1)='+'|SUBSTR(CARD.KEY,1,1)='-' THEN GOTO CH;
41:     READ FILE(CNST) INTO(T1) KEY(CARD.KEY);
42:     T1.TRUE=CARD.TRUE;
43:     T1.DTE=DATES;
44:     REWRITE FILE(CNST) FROM(T1);
45:     GOTO LOOP;
46: CH:   /* TRUE MILEAGE ALTERING RECORD EDIT */
47:     IF SUBSTR(C,4,1)¬='.'|SUBSTR(C,8,1)¬=','|SUBSTR(C,13,1)¬=','|
48:     (SUBSTR(C,17,1)¬=' ' & SUBSTR(C,17,1)¬=',') THEN DO;
49:       PRINTER='**ERROR IN CODING OF , . OR BLANKS IN CARD COLUMNS'||
50:               ' 4, 8, 13, 17, CHANGES IN TRUE MILEAGE NOT MADE';
51:       CALL PRINTX(2);
52:       GOTO LOOP;
53:       END;
54:    IF SUBSTR(C,9,1)¬='I'&SUBSTR(C,9,1)¬='P'&SUBSTR(C,9,1)¬='S' THEN
55:         DO;
56:       PRINTER='**ERROR IN SYSTEM CODE, CARD COLUMN 9, MUST BE'||
57:               ' I, P, OR S  CHANGES IN TRUE MILEAGE NOT MADE';
58:       CALL PRINTX(2);
```

```
/**MEMBER NAME PDMR**/

59:         GOTO LOOP;
60:         END;
61:      READ FILE (CNST) INTO (T1) KEY (SUBSTR(C,9,4)||SUBSTR(C,14,3));
62: COMP:  /* TRUE MILEAGE CHANGE ROUTINE */
63:       T1.DTE=DATES;
64:       T1.TRUE=T1.TRUE+NUMBER;
65:       REWRITE FILE(CNST) FROM(T1);
66:      PRINTER = (10)' ' || T1.KEY || T1.TRUE;
67:      CALL PRINTX (1);
68:       READ FILE(CNST) INTO(T1);
69:       IF SUBSTR(T1.KEY,2,3)¬=SUBSTR(C,10,3) THEN GOTO LOOP;
70:       IF SUBSTR(T1.KEY,5,3)>SUBSTR(C,18,3) & SUBSTR(C,18,3)¬=' ' THEN
71:         GOTO LOOP;
72:       ELSE GOTO COMP;
73: FINISH:
74:       CLOSE FILE(CNST);
75:       CLOSE FILE(CD);
76:       CALL EXIT (PARM);
77: END REVISE;
```

COPY (True Mileage file) --

Member Name	PBM
Language	PL/I
Subroutines	PRINTX1
Files	SYSPRINT -- IBM messages PRINTER -- COPY output TRUMILE -- True Mileage file SAVETRM -- Backup copy (output)
Instruction	1 - 3 "PBM"

COPY prepares a backup copy of the True Mileage file. The backup copy is a sequential version of the file, with identical record length (16). A dummy record containing the date is first written. This record is followed by the True Mileage records. The records are counted during the COPY operation, and the number of records in the file printed after the last record is rewritten.

The PBM program listing follows:

```
/* :COPY,FILE=TRUMILE */

 1: /* :COPY,FILE=TRUMILE */

 2: COPY:  PROCEDURE (PARM) OPTIONS (MAIN);

 3: /* INSTRUCTION */
 4: DECLARE
 5:    INSTR CHAR(80) EXT,
 6:    #_HDGS PIC'Z' DEF INSTR POS(72);

 7: /* PRINT ROUTINE */
 8: DECLARE
 9:    PARM CHAR(100),
10:   (HEADING(9),PRINTER) CHAR(132) EXT,
11:    PRINTX ENTRY (PIC'Z');

12: /* FILES */
13: DECLARE
14:    RECORD CHAR(16) BASED (PTR),
15:    BACKDD CHAR(8) STATIC INIT ('SAVETRM'),
16:    PERMDD CHAR(8) STATIC INIT ('TRUMILE'),
17:    PERM FILE RECORD KEYED ENV (INDEXED),
18:    BACKUP FILE RECORD;

19: /* OTHER VARIABLES */
20: DECLARE
21:    UD CHAR(6),
22:    CNTR BIN FIXED (31),
23:    PCNTR PIC'ZZZZZ9';


24: /***** INITIALIZATION *****/

25:    CALL INIT (PARM);

26:    /* SET UP HEADINGS */
27:    #_HDGS = 2;
28:    HEADING(1) = PERMDD || 'FILE COPY ROUTINE';

29:    /* INIT FILES */
30:    OPEN FILE (PERM) INPUT TITLE (PERMDD);
31:    OPEN FILE (BACKUP) OUTPUT TITLE (BACKDD);
32:    ON ENDFILE (PERM) GOTO DONE;

33:    /* RECORD DATE */
34:    UD = DATE;
35:    PTR = ADDR(HEADING(9));
36:    RECORD = SUBSTR(UD,3,2) || '/' ||
37:         SUBSTR(UD,5,2) || '/' || SUBSTR(UD,1,2);
38:    WRITE FILE (BACKUP) FROM (RECORD);


39: /***** MAIN EXECUTION LOOP *****/

40:    DO CNTR=1 TO 999999;
41:       READ FILE (PERM) SET (PTR);
42:       WRITE FILE (BACKUP) FROM (RECORD);
43:       END;
```

```
/* :COPY,FILE=TRUMILE */

 44: DONE:
 45:    PCNTR = CNTR - 1;
 46:    PRINTER = 'NUMBER OF RECORDS IN FILE:  ' || PCNTR;
 47:    CALL PRINTX (3);
 48:    CLOSE FILE (PERM);
 49:    CLOSE FILE (BACKUP);
 50:    CALL EXIT (PARM);

 51: END COPY;
```

CREATE (True Mileage file) --

Member Name	PAM
Language	PL/I
Subroutines	SYSPRINT -- IBM messages PRINTER -- CREATE output TRUMILE -- True Mileage file (output) SAVETRM -- Backup copy
Instruction	1 - 3 "PAM"

CREATE restores the True Mileage file from a backup copy saved via program COPY. The first record in the file is a dummy record, containing the date on which the file was copied. This date is printed prior to performing the create operation. After printing the date, the records are read from the backup copy and written into the True Mileage file, destroying the previous file. As with COPY, the records are counted as they are written. The count is printed after the create operation is complete.

The PAM program listing follows:

```
/* :CREATE,FILE=TRUMILE */

  1: /* :CREATE,FILE=TRUMILE */

  2: CREATE:  PROCEDURE (PARM) OPTIONS (MAIN);

  3: /* INSTRUCTION */
  4: DECLARE
  5:    INSTR CHAR(80) EXT,
  6:    #_HDGS PIC'Z' DEF INSTR POS(72);

  7: /* PRINT ROUTINE */
  8: DECLARE
  9:    PARM CHAR(100),
 10:   (HEADING(9),PRINTER) CHAR(132) EXT,
 11:    PRINTX ENTRY (PIC'Z');

 12: /* FILES */
 13: DECLARE
 14:    RECORD CHAR(16) BASED (PTR),
 15:    BACKDD CHAR(8) STATIC INIT ('SAVETRM'),
 16:    PERMDD CHAR(8) STATIC INIT ('TRUMILE'),
 17:    PERM FILE RECORD KEYED ENV (INDEXED),
 18:    BACKUP FILE RECORD;

 19: /* OTHER VARIABLES */
 20: DECLARE
 21:    CNTR BIN FIXED (31),
 22:    PCNTR PIC'ZZZZZ9';


 23: /***** INITIALIZATION *****/

 24:    CALL INIT (PARM);

 25:    /* SET UP HEADINGS */
 26:    #_HDGS = 2;
 27:    HEADING(1) = PERMDD || 'FILE CREATION ROUTINE';

 28:    /* INIT FILES */
 29:    OPEN FILE (BACKUP) INPUT TITLE (BACKDD);
 30:    OPEN FILE (PERM) OUTPUT TITLE (PERMDD);
 31:    ON ENDFILE (BACKUP) GOTO DONE;

 32:    /* PRINT DATE */
 33:    READ FILE (BACKUP) SET (PTR);
 34:    PRINTER = '   DATE OF BACKUP FILE IS ' || RECORD;
 35:    CALL PRINTX (1);
 36:    PRINTER = ' ';
 37:    CALL PRINTX (1);


 38: /***** MAIN EXECUTION LOOP *****/

 39:    DO CNTR=1 TO 999999;
 40:       READ FILE (BACKUP) SET (PTR);
 41:       WRITE FILE (PERM) FROM (RECORD) KEYFROM (SUBSTR(RECORD,2));
 42:       END;

 43: DONE:
```

```
/* :CREATE,FILE=TRUMILE */

 44:     PCNTR = CNTR - 1;
 45:     PRINTER = 'NUMBER OF RECORDS IN FILE:  ' || PCNTR;
 46:     CALL PRINTX (3);
 47:     CLOSE FILE (PERM);
 48:     CLOSE FILE (BACKUP);
 49:     CALL EXIT (PARM);

 50: END CREATE;
```

CREATE-TRAFSUB --

Member Name	CRT
Language	PL/I
Subroutines	none
Files	SYSPRINT -- Printer output TRAFFIC -- Traffic file TRUMILE -- True Mileage file TRAFSUB -- Traffic Summary file (output)
Instruction	1 - 3 "CRT"

CREATE-TRAFSUB creates the Traffic Summary file utilizing the Traffic and True Mileage files. Vehicle miles and section lengths are included in the file for three years (from the first three data year positions of the Traffic file). The section length is obtained through use of the True Mileage file. For each section, the program must obtain the ADT's from the Traffic file, and weight these over the section (each section may have any number of minor section breaks). The program checks for a valid sequence of remark codes in the Traffic file. For example, an "M" record (municipal minor section break) cannot follow a "W" record (rural major section break). If such a sequence, or any other invalid sequence, is detected, an error message is printed. When a record contains no data for one or more of the years, no vehicle miles will be stored in the file for those years in the section the record is located. Hence, major section break records beginning and ending a section may contain data pertaining to the sections preceding and following the section, but a minor break record within the section containing no data will cause that data to be ignored in the section. On rural records, counts are taken each year. Hence, most records contain data for all three years. However, when a new road is built, only the most recent year will be coded; the following year, only two years will be coded. It is not valid, however, to have any other combination of uncoded years on rural records (such as only the second year coded, or no years at all coded). The presence of such a record will cause a message to be generated. Any combination of blank data may be present in municipal sections; counts are not necessarily taken every year. Hence, no error messages are generated pertaining to blank data in municipal sections. No data is coded in non-existent sections. If any is present, it is not used.

The CRT program listing follows:

```
/* :CREATE_TRAFSUB */

 1: /* :CREATE_TRAFSUB */

 2: CREATE:  PROCEDURE OPTIONS (MAIN);

 3: /* TRAFFIC FILE */
 4: DECLARE
 5:    1  TRF BASED (PTR_TRF),
 6:       2  DELETE CHAR(1),
 7:       2  KEY CHAR(13),
 8:       2 (RT_#,MPOST) DEC FIXED (3,0),
 9:       2  OFFSET DEC FIXED (5,3),
10:       2 (ACT_EST,REMARK) CHAR(1),
11:       2  D(3),
12:          3  YR DEC FIXED (3,0),
13:          3  ADT DEC FIXED (5,0),
14:          3  PER(3) DEC FIXED (3,3),
15:    TRFB CHAR(80) BASED (PTR_TRF),
16:    HOLD_TRF CHAR(33),
17:    1  SAVE BASED (PTR_SAVE) LIKE TRF,
18:    SAVE_B CHAR(80) STATIC,
19:    TRAFFIC FILE INT RECORD KEYED ENV (INDEXED);

20: /* TRUE MILEAGE FILE */
21: DECLARE
22:    1  TRM BASED (PTR_TRM),
23:       2  KEY CHAR(5),
24:       2  MPOST PIC'999',
25:       2  TRUE DEC FIXED (7,3),
26:    MLGE(0:999) DEC FIXED (7,3) STATIC,
27:    TRUMILE FILE INT RECORD KEYED ENV (INDEXED GENKEY);

28: /* TRAFSUB FILE */
29: DECLARE
30:    1  SUB STATIC,
31:       2  DUMMY1 CHAR(1) INIT (' '),
32:       2  KEY CHAR(13),
33:       2  REMARK CHAR(1),
34:       2  LENGTH DEC FIXED (7,3),
35:       2  D(3),
36:          3  VM(4) DEC FIXED (11,3),
37:       2  DUMMY2 CHAR(5) INIT (' '),
38:    TRAFSUB FILE INT RECORD OUTPUT KEYED ENV (INDEXED);

39: /* OTHER VARIABLES */
40: DECLARE
41:   (C1,C2) CHAR(1),
42:   (ADT(3,4),VM(3,4),TOT1_VM(3,4),TOT2_VM(3,4))
43:       DEC FIXED (11,3) STATIC,
44:   (STOT1_VM(3,4),STOT2_VM(3,4)) DEC FIXED (11,3) STATIC,
45:   (TOT1_LENGTH,TOT2_LENGTH) DEC FIXED (9,3),
46:   (STOT1_LENGTH,STOT2_LENGTH) DEC FIXED (9,3),
47:   (X1,X2) DEC FIXED (7,3),
48:   (FLAG1(3),FLAG2(3)) CHAR(1) STATIC,
49:    LENGTH DEC FIXED (5,3) STATIC,
50:    SAVE_SYSTEM CHAR(1) STATIC INIT (' '),
51:    YR_RCD CHAR(96),
52:    ERROR DEC FIXED (3,0),
53:    ERR_FLAG CHAR(1) STATIC INIT (' ');
```

```
/* :CREATE_TRAFSUB */


54: /***** INITIALIZATION *****/

55:     OPEN
56:        FILE (TRAFFIC),
57:        FILE (TRUMILE),
58:        FILE (TRAFSUB);
59:     ON ENDFILE (TRAFFIC) TRF.RT_# = 0;
60:     ON ENDFILE (TRUMILE) TRM.KEY = ' ';
61:     ON KEY (TRAFSUB) BEGIN;
62:        PUT FILE (SYSPRINT) SKIP EDIT ('***KEY ERR:  ',SUB.KEY) (A);
63:        GOTO ENDUP;
64:        END;

65:     /* READ FIRST TRAFFIC RECORD */
66:     READ FILE (TRAFFIC) SET (PTR_TRF);
67:     YR_RCD = TRFB;
68:     WRITE FILE (TRAFSUB) FROM (YR_RCD) KEYFROM (SUBSTR(YR_RCD,2));

69:     /* READ FIRST INTERSTATE RECORD */
70:     READ FILE (TRAFFIC) SET (PTR_TRF);
71:     PTR_SAVE = ADDR(SAVE_B);

72:     /* INIT VAR */

73: INIT_VAR:
74:     VM = 0;
75:     TOT1_VM, TOT2_VM = 0;
76:     TOT1_LENGTH, TOT2_LENGTH = 0;
77:     IF SUBSTR(TRF.KEY,1,1)¬=SAVE_SYSTEM THEN DO;
78:        STOT1_VM, STOT2_VM = 0;
79:        STOT1_LENGTH, STOT2_LENGTH = 0;
80:        END;
81:     READ FILE (TRUMILE) SET (PTR_TRM) KEY (SUBSTR(TRF.KEY,1,4));
82:     MLGE = 0;
83:     DO WHILE (SUBSTR(TRM.KEY,2,4)=SUBSTR(TRF.KEY,1,4));
84:        MLGE(TRM.MPOST) = TRM.TRUE;
85:        READ FILE (TRUMILE) SET (PTR_TRM);
86:        END;
87:     X2 = 0;


88: /***** EXECUTION LOOP *****/

89: LOOP:

90:     /* SAVE PREVIOUS VALUES */
91:     SAVE_SYSTEM = SUBSTR(TRF.KEY,1,1);
92:     SAVE_B = TRFB;
93:     X1 = X2;

94: LOOPE:

95:     /* GET NEXT TRAFFIC RECORD */
96:     READ FILE (TRAFFIC) SET (PTR_TRF);

97:     /* CHECK FOR NEW ROUTE */
```

```
/* :CREATE_TRAFSUB */

 98:     IF TRF.RT_#¬=SAVE.RT_# THEN DO;
 99:        SUB.KEY = SAVE.KEY;
100:        SUB.REMARK = 'E';
101:        SUB.LENGTH = 0;
102:        SUB.D = 0;
103:        WRITE FILE (TRAFSUB) FROM (SUB) KEYFROM (SUB.KEY);
104:        SUBSTR(SUB.KEY,5) = '999RURAL';
105:        SUB.LENGTH = TOT2_LENGTH;
106:        DO I=1 TO 3;
107:           SUB.D(I).VM(*) = TOT2_VM(I,*);
108:           END;
109:        WRITE FILE (TRAFSUB) FROM (SUB) KEYFROM (SUB.KEY);
110:        STOT1_VM = STOT1_VM + TOT1_VM;
111:        STOT2_VM = STOT2_VM + TOT2_VM;
112:        STOT1_LENGTH = STOT1_LENGTH + TOT1_LENGTH;
113:        STOT2_LENGTH = STOT2_LENGTH + TOT2_LENGTH;
114:        IF SUBSTR(TRF.KEY,1,1)¬=SAVE_SYSTEM THEN DO;
115:           SUBSTR(SUB.KEY,2) = '999RURAL';
116:           SUB.LENGTH = STOT2_LENGTH;
117:           DO I=1 TO 3;
118:             SUB.D(I).VM(*) = STOT2_VM(I,*);
119:              END;
120:           WRITE FILE (TRAFSUB) FROM (SUB) KEYFROM (SUB.KEY);
121:           END;
122:        IF TRF.RT_#¬=0 THEN GOTO INIT_VAR;
123:        GOTO ENDUP;
124:        END;

125:     /* VALID REMARK SEQUENCES:

126:           WW  OW  TW  NW  RW  MW  SW  LW  CW
127:           WO  OO  TO  NO  RO  MO  SO  LO  CO
128:           WT  OT  TT  NT  RT  MT  ST  LT  CT
129:           WN  ON  TN  NN  RN  MN  SN  LN  CN
130:           WR  OR          RR
131:                   TM          MM
132:           WS  OS  TS  NS                  CS
133:           WL  OL  TL  NL                  CL
134:           WC  OC  TC  NC          SC  LC  CC
                                                   */
135:
136:     C1 = SAVE.REMARK;
137:     C2 = TRF.REMARK;
138:     IF C2='W' | C2='O' | C2='T' | C2='N' THEN GOTO VALID;
139:     IF C2='R' & (C1='W' | C1='O' | C1='R') THEN GOTO VALID;
140:     IF C2='M' & (C1='T' | C1='M') THEN GOTO VALID;
141:     IF (C2='S' | C2='L' | C2='C') &
142:        (C1='W' | C1='O' | C1='T' | C1='N' | C1='C')
143:        THEN GOTO VALID;
144:     IF C2='C' & (C1='S' | C1='L') THEN GOTO VALID;
145:     ERROR = 1;
146:     GOTO ERROR_MSG;

147: VALID:

148:     /* SEQUENCES THAT DO NOT REQUIRE PROCESSING:

149:           SW  LW  CW
150:           ST  LT  CT
```

```
/* :CREATE_TRAFSUB */

151:          SO  LO  CO
152:          SN  LN  CN
153:                      */
154:     IF (C1='S' | C1='L' | C1='C') &
155:        (C2='W' | C2='O' | C2='T' | C2='N')
156:        THEN DO;
157:        X2 = MLGE(TRF.MPOST) + TRF.OFFSET;
158:        GOTO LOOP;
159:        END;

160:     /* SEQUENCES REQUIRING DUMMY RECORDS:

161:          WS  TS  CS  NS  OS
162:          WL  TL  CL  NL  OL
163:          WC  TC  CC  NC  OC  SC  LC
164:                                      */
165:     IF C2='S' | C2='L' | C2='C' THEN DO;
166:        SUB.LENGTH = 0;
167:        SUB.D = 0;
168:        IF C1¬='C' & C1¬='S' & C1¬='L' THEN DO;
169:           SUB.KEY = SAVE.KEY;
170:           SUB.REMARK = 'D';
171:           WRITE FILE (TRAFSUB) FROM (SUB) KEYFROM (SUB.KEY);
172:           END;
173:        SUB.KEY = TRF.KEY;
174:        SUB.REMARK = C2;
175:        WRITE FILE (TRAFSUB) FROM (SUB) KEYFROM (SUB.KEY);
176:        GOTO LOOP;
177:        END;

178:     /* NON-EXISTANT SECTIONS (NN,NW,NT,NO) */
179:     IF C1='N' THEN DO;
180:        SUB.KEY = SAVE.KEY;
181:        SUB.REMARK = 'N';
182:        X2 = MLGE(TRF.MPOST) + TRF.OFFSET;
183:        SUB.LENGTH = X2 - X1;
184:        SUB.D = 0;
185:        WRITE FILE (TRAFSUB) FROM (SUB) KEYFROM (SUB.KEY);
186:        TOT2_LENGTH = TOT2_LENGTH + SUB.LENGTH;
187:        GOTO LOOP;
188:        END;

189:     /* ALL REMAINING SEQUENCES REQUIRE COMPUTATIONS */

190:     /* INIT VAR IF NEW SECTION */
191:     IF C1='W' | C1='T' | C1='O' THEN DO;
192:        SUB.REMARK = C1;
193:        SUB.LENGTH = 0;
194:        SUB.KEY = SAVE.KEY;
195:        VM = 0;
196:        FLAG2 = 'X';
197:        END;

198:     /* CALCULATE SECTION LENGTH */
199:     X2 = MLGE(TRF.MPOST) + TRF.OFFSET;
200:     LENGTH = X2 - X1;
201:     SUB.LENGTH = SUB.LENGTH + (X2-X1);
```

```
/* :CREATE_TRAFSUB */

202:     /* IF JCT, CARRY ORIGINAL DATA THROUGH */
203:     IF LENGTH<=.01 & (C1='R' | C1='M' | C2='R' | C2='M')
204:        THEN DO;
205:           HOLD_TRF = SUBSTR(TRFB,24,33);
206:           SUBSTR(TRFB,24,33) = SUBSTR(SAVE_B,24,33);
207:           END;

208:     /* CHECK FOR BLANK DATA */
209:     FLAG1 = 'X';
210:     DO I=1 TO 3;
211:        IF TRF.D(I).YR=0 | SAVE.D(I).YR=0 THEN FLAG1(I) = 'B';
212:        END;

213:     /* VALID BLANK/NON-BLANK COMBINATIONS ON RURAL SECTIONS:

214:           B  B  X  (NEW ROAD)
215:           B  X  X  (NEW ROAD--USED TWO YEARS)
216:           X  X  X  (ALL YEARS CODED)
217:                                    */
218:     IF C1='T' | C1='M' THEN GOTO ALL_RIGHT;
219:     IF FLAG1(1)='X' & FLAG1(2)='X' & FLAG1(3)='X' |
220:        FLAG1(1)='B' & FLAG1(2)='X' & FLAG1(3)='X' |
221:        FLAG1(1)='B' & FLAG1(2)='B' & FLAG1(3)='X'
222:        THEN GOTO ALL_RIGHT;
223:     ERROR = 2;
224:     GOTO ERROR_MSG;

225: ALL_RIGHT:
226:     DO I=1 TO 3;
227:        IF FLAG1(I)='B' THEN FLAG2(I) = 'B';
228:        END;

229:     /* CALCULATE ADT'S */
230:     ADT(*,1) = (SAVE.ADT + TRF.ADT) / 2;
231:     DO I=1 TO 3;
232:        ADT(*,I+1) = (SAVE.ADT*SAVE.PER(*,I) + TRF.ADT*TRF.PER(*,I)) / 2
233:        END;

234:     /* CALCULATE VEHICLE MILES */
235:     VM = VM + LENGTH*ADT;

236:     /* RESTOR DATA IF AT JCT */
237:     IF LENGTH<=.01 & (C1='R' | C1='M' | C2='R' | C2='M')
238:        THEN SUBSTR(TRFB,24,33) = HOLD_TRF;

239:     /* IF NOT SECTN BREAK, RETURN TO LOOP */
240:     IF C2='R' | C2='M' THEN GOTO LOOP;

241:     /* ZERO OUT VM'S IN SECTIONS WITH BLANK FIELDS */
242:     DO I=1 TO 3;
243:        IF FLAG2(I)='B' THEN VM(I,*) = 0;
244:        END;

245:     /* ZERO OUT COMM, PICKUPS, OUT OF STATE IF MUNICIPAL */
246:     IF SUB.REMARK='T' THEN DO I = 2 TO 4;
247:        VM(*,I) = 0;
248:        END;
```

```
/* :CREATE_TRAFSUB */

249:     /* WRITE TO OUTPUT FILE */
250:     DO I=1 TO 3;
251:        SUB.D(I).VM(*) = VM(I,*);
252:        END;
253:     WRITE FILE (TRAFSUB) FROM (SUB) KEYFROM (SUB.KEY);
254:     IF SUB.REMARK='T'
255:       THEN DO;
256:          TOT1_VM = TOT1_VM + VM;
257:          TOT1_LENGTH = TOT1_LENGTH + SUB.LENGTH;
258:          END;
259:        ELSE DO;
260:          TOT2_VM = TOT2_VM + VM;
261:          TOT2_LENGTH = TOT2_LENGTH + SUB.LENGTH;
262:          END;
263:     GOTO LOOP;

264: ERROR_MSG:
265:     IF ERR_FLAG=' ' THEN DO;
266:        PUT FILE (SYSPRINT) SKIP EDIT
267:           ('    CREATE_TRAFSUB DIAGNOSTICS') (A);
268:        ERR_FLAG = 'X';
269:        END;
270:     IF ERROR = 1
271:        THEN PUT FILE (SYSPRINT) SKIP EDIT
272:        ('REMARK SEQUENCE ERROR (',C1,C2,') AT KEYS ',SAVE.KEY,' AND ',
273:        TRF.KEY) (A);
274:     IF ERROR=2
275:        THEN PUT FILE (SYSPRINT) SKIP EDIT
276:        ('INVALID BLANK DATA ON RURAL RECORD ',SAVE.KEY,' OR ',
277:        TRF.KEY) (A);
278:     IF ERROR=1 THEN GOTO LOOPE;
279:     GOTO LOOP;

280: ENDUP:
281:     PUT FILE (SYSPRINT) SKIP (3) EDIT
282:        ('TRAFSUB FILE CREATED') (A);
283:     CLOSE
284:        FILE (TRAFFIC),
285:        FILE (TRUMILE),
286:        FILE (TRAFSUB);

287: END CREATE;
```

LIST-TRAFSUB --

Member Name	LST	
Language	PL/I	
Subroutines	PRINTX1	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- LIST-TRAFSUB output
	TRAFSUB	-- Traffic Summary file
Instruction	1 - 3	"LST"
	40 - 43	Beginning route number
	56 - 59	Ending route number

LIST-TRAFSUB provides a listing of the Traffic Summary file. For each of the three data years is shown the vehicle miles for all vehicles, commercial vehicles, and out-of-state vehicles (vehicle miles for pickups is not shown). The section length, key, and remark are also printed.

The LST program listing follows:

```
/* LIST-TRAFSUB,DATA=XXXXX */

 1: /* LIST-TRAFSUB,DATA=XXXXX */

 2: LIST:  PROCEDURE (PARM) OPTIONS (MAIN);

 3: /* INSTRUCTION & PRINT ROUTINE */
 4: DECLARE
 5:    PARM CHAR(100),
 6:    INSTR CHAR(80) EXT,
 7:    STARTKEY CHAR(4) DEF INSTR POS(40),
 8:    ENDKEY CHAR(4) DEF INSTR POS(56),
 9:    #_HDGS PIC'Z' DEF INSTR POS(72),
10:   (HEADING(9),PRINTER) CHAR(132) EXT,
11:    PRINTX ENTRY (PIC'Z');

12: /* TRAFSUB FILE */
13: DECLARE
14:    1  SUB BASED (PTR_SUB),
15:       2  DUM1 CHAR(1),
16:       2  RT_# CHAR(4),
17:       2  MPOST CHAR(3),
18:       2  OFFSET CHAR(6),
19:       2  REMARK CHAR(1),
20:       2  LENGTH DEC FIXED (7,3),
21:       2  DATA(3),
22:          3 (VM1,VM2,VM3,VM4) DEC FIXED (11,3),
23:    TRAFSUB FILE INT RECORD KEYED ENV (INDEXED GENKEY);

24: /* OUTPUT STRUCTURE */
25: DECLARE
26:    OUT CHAR(132) STATIC INIT (' '),
27:    1  O DEF OUT,
28:       2  RT_# CHAR(5),
29:       2  MPOST CHAR(3),
30:       2  OFFSET CHAR(7),
31:       2  REMARK CHAR(1),
32:       2  LENGTH PIC'ZZZZZV.ZZZ',
33:       2  DATA(3),
34:          3 (VM1,VM2,VM4) PIC'(10)Z';


35: /***** INITIALIZATION *****/

36:    CALL INIT (PARM);
37:    #_HDGS = 2;
38:    HEADING(1) = 'RT-#  MP OFFSET    LENGTH  ***********1******' ||
39:       '*********  ************2**************' ||
40:       '  ************3**************';
41:    OPEN FILE (TRAFSUB);
42:    ON ENDFILE (TRAFSUB) BEGIN;
43:       PRINTER = '   END OF FILE.';
44:       CALL PRINTX (3);
45:       GOTO DONE;
46:       END;
47:    READ FILE (TRAFSUB) SET (PTR_SUB) KEY (STARTKEY);


48: /***** EXECUTION LOOP *****/
```

```
/* LIST-TRAFSUB,DATA=XXXXX */

49:     DO WHILE (SUB.RT_#<=ENDKEY);
50:        O = SUB, BY NAME;
51:        PRINTER = OUT;
52:        CALL PRINTX (1);
53:        READ FILE (TRAFSUB) SET (PTR_SUB);
54:        END;

55: DONE:
56:     CLOSE FILE (TRAFSUB);
57:     CALL EXIT (PARM);

58: END LIST;
```

TRAFFIC-BY-SECTIONS --

Member Name	TRT	
Language	PL/I	
Subroutines	PRINTX1	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- TRAFFIC-BY-SECTIONS output
	ROADLOG	-- Roadlog file
	TRAFSUB	-- Traffic Summary file
	CNTYTBL	-- Table of county names
Instruction	1 - 3	"TRT"
	40 - 43	Beginning route number
	56 - 59	Ending route number

TRT produces the main body of the Traffic by Sections report. It utilizes the Traffic Summary and Roadlog files. It must also read the county table in HIS.TABLES in order to print the county names. The vehicle mileage and section length are obtained from the Traffic Summary file. From this, the ADT's are calculated (vehicle miles/length). The Roadlog file is accessed to obtain the section descriptions and the county numbers. The county names are read into an array during program initialization, and the county numbers are used as indexes into the array to obtain the county name.

The TRT program listing follows:

```
/* :TRAFFIC-BY-SECTIONS,DATA=XXXXX */

 1: /* :TRAFFIC-BY-SECTIONS,DATA=XXXXX */

 2: TRBYSEC:  PROCEDURE (PARM) OPTIONS (MAIN);

 3: /* INSTRUCTION AND PRINT ROUTINE */
 4: DECLARE
 5:    PARM CHAR(100),
 6:    INSTR CHAR(80) EXT,
 7:    PAGE_SIZE PIC'ZZ' DEF INSTR POS(7),
 8:    PAGE_POSITION PIC'ZZ' DEF INSTR POS(9),
 9:    STARTKEY CHAR(4) DEF INSTR POS(40),
10:    ENDKEY CHAR(4) DEF INSTR POS(56),
11:    #_HDGS PIC'Z' DEF INSTR POS(72),
12:   (HEADING(9),PRINTER) CHAR(132) EXT,
13:    PRINTX ENTRY (PIC'Z'),
14:    PRINTXA ENTRY (PIC'Z',PIC'ZZ');

15: /* ROADLOG VARIABLES */
16: DECLARE
17:    1  RLG BASED (PTR_RLG),
18:       2  DUM1 CHAR(14),
19:       2  REMARK CHAR(2),
20:       2  DUM2 CHAR(14),
21:       2  DESCR CHAR(35),
22:       2  DUM3 CHAR(16),
23:       2  COUNTY_# DEC FIXED (3,0),
24:    ROADLOG FILE INT RECORD KEYED ENV (INDEXED);

25: /* TRAFFIC SUMMARY FILE */
26: DECLARE
27:    1  SUB BASED (PTR_SUB),
28:       2  DUM1 CHAR(1),
29:       2  SYSTEM CHAR(1),
30:       2  RT_# PIC'999',
31:       2  MPOST PIC'999',
32:       2  DUM2 CHAR(2),
33:       2  OFFSET CHAR(3),
34:       2  DUM3 CHAR(1),
35:       2  REMARK CHAR(1),
36:       2  LENGTH DEC FIXED (7,3),
37:       2  D(3),
38:          3 (VM,VM_FOR,VM_PIC,VM_COMM) DEC FIXED (11,3),
39:    1  SUB2 BASED (PTR_SUB),
40:       2  DUM1 CHAR(1),
41:       2  KEY CHAR(13),
42:       2  YR(3) CHAR(2),
43:    TRAFSUB FILE INT RECORD KEYED ENV (INDEXED GENKEY);

44: /* OTHER VARIABLES */
45: DECLARE
46:    LINES CHAR(103) STATIC INIT ((103)'-'),
47:    COUNTY(0:57) CHAR(15) STATIC,
48:    RECORD CHAR(80) BASED (PTR_TBL),
49:    CNTYTBL FILE INT RECORD,
50:    ZRT_# PIC'ZZZZ',
51:    ZMPOST PIC'ZZ9',
52:    LNGTH PIC'ZZ9V.999',
53:    ADT(3,3) DEC FIXED (7,0),
```

```
/* :TRAFFIC-BY-SECTIONS,DATA=XXXXX */

 54:   (FADT(3,3),VM(3)) PIC'(6)Z';


 55: /***** INITIALIZATION *****/

 56:    CALL INIT (PARM);
 57:    #_HDGS = 5;
 58:    HEADING(3) = '  MILE';
 59:    SUBSTR(HEADING(3),71,31) = 'SECTION *******ADT*******   VEH';
 60:    HEADING(4) = '  POST        SECTION DESCRIPTION          ' ||
 61:       'COUNTY        TRAFFIC TYPE  LENGTH                    MILES';

 62:    /* READ TABLE OF COUNTY NAMES */
 63:    COUNTY(0) = '*** INVALID ***';
 64:    COUNTY(57) = ' OUT OF STATE';
 65:    OPEN FILE (CNTYTBL);
 66:    DO I=1 TO 56;
 67:       READ FILE (CNTYTBL) SET (PTR_TBL);
 68:       COUNTY(I) = SUBSTR(RECORD,16,15);
 69:       END;
 70:    CLOSE FILE (CNTYTBL);

 71:    /* INIT FILES */
 72:    OPEN
 73:       FILE (TRAFSUB),
 74:       FILE (ROADLOG);
 75:    ON ENDFILE (TRAFSUB) GOTO DONE;
 76:    ON KEY (ROADLOG) BEGIN;
 77:       RLG.DESCR = '***** NO ROADLOG RECORD *****';
 78:       RLG.COUNTY_# = 0;
 79:       END;

 80:    /* GET DATES */
 81:    READ FILE (TRAFSUB) SET (PTR_SUB);
 82:    SUBSTR(HEADING(4),81,2) = SUB2.YR(1);
 83:    SUBSTR(HEADING(4),87,2) = SUB2.YR(2);
 84:    SUBSTR(HEADING(4),93,2) = SUB2.YR(3);

 85:    /* READ FIRST RECORD OF ROUTE */
 86:    READ FILE (TRAFSUB) SET (PTR_SUB) KEY (STARTKEY);
 87:    ZRT_# = SUB.RT_#;
 88:    HEADING(1) = '          FA' || SUB.SYSTEM || ZRT_#;


 89: /***** EXECUTION LOOP *****/

 90: LOOP:
 91:    READ FILE (ROADLOG) SET (PTR_RLG) KEY (SUB2.KEY);
 92:    IF SUB.REMARK='C' | SUB.REMARK='S' | SUB.REMARK='L' THEN DO;
 93:       PRINTER = (35)' ' || RLG.DESCR;
 94:       /* TEMP */ IF SUB.REMARK='C' THEN SUBSTR(PRINTER,54,17) =
 95:          SUBSTR(DESCR,21,3) || ' TO ' || SUBSTR(DESCR,27,3) ||
 96:          SUBSTR(DESCR,31,3);
 97:       CALL PRINTXA (2,8);
 98:       GOTO NEXT_RECORD;
 99:       END;
100:    ZMPOST = SUB.MPOST;
101:    PRINTER = ZMPOST || SUB.OFFSET || '  ' || RLG.DESCR;
```

```
/* :TRAFFIC-BY-SECTIONS,DATA=XXXXX */

102:    CALL PRINTX (1);
103:    IF SUB.REMARK='D' THEN GOTO NEXT_RECORD;
104:    IF SUB.REMARK='E' THEN DO;
105:       DO WHILE (SUB.REMARK='E');
106:          READ FILE (TRAFSUB) SET (PTR_SUB);
107:          END;
108:       IF SUBSTR(SUB2.KEY,1,4)>ENDKEY THEN GOTO DONE;
109:       IF PAGE_SIZE-PAGE_POSITION>2 THEN DO;
110:          PRINTER = LINES;
111:          CALL PRINTX (2);
112:          END;
113:       ZRT_# = SUB.RT_#;
114:       HEADING(1) = '          FA' || SUB.SYSTEM || ZRT_#;
115:       IF PAGE_POSITION+10<PAGE_SIZE
116:          THEN DO;
117:             PRINTER = HEADING(1);
118:             CALL PRINTX (3);
119:             PRINTER = ' ';
120:             CALL PRINTX (2);
121:             END;
122:          ELSE PAGE_POSITION = PAGE_SIZE;
123:       GOTO LOOP;
124:       END;
125:    IF PAGE_SIZE-PAGE_POSITION<=4 THEN CALL PRINTX (9);
126:    LNGTH = SUB.LENGTH;
127:    IF SUB.REMARK='N' THEN DO;
128:       PRINTER = (46)' ' || '** NON EXISTANT**' || (8)' ' || LNGTH;
129:       CALL PRINTX (2);
130:       PRINTER = ' ';
131:       CALL PRINTX (1);
132:       GOTO NEXT_RECORD;
133:       END;
134:    ADT(*,1) = SUB.D.VM / SUB.LENGTH;
135:    ADT(*,2) = SUB.D.VM_COMM / SUB.LENGTH;
136:    ADT(*,3) = SUB.D.VM_FOR / SUB.LENGTH;
137:    FADT = ADT;
138:    VM(1) = SUB.D(3).VM;
139:    VM(2) = SUB.D(3).VM_COMM;
140:    VM(3) = SUB.D(3).VM_FOR;

141:    /* ALL VEHICLES */
142:    IF ADT(1,1)+ADT(2,1)+ADT(3,1)¬=0
143:       THEN PRINTER = (57)' ' || 'ALL VEHICLES          ' ||
144:          FADT(1,1) || FADT(2,1) || FADT(3,1) || VM(1);
145:       ELSE PRINTER = ' ';
146:    CALL PRINTX (1);

147:    /* COMMERCIAL VEHICLES */
148:    IF RLG.REMARK='OS' THEN RLG.COUNTY_# = 57;
149:    IF ADT(1,2)+ADT(2,2)+ADT(3,2)¬=0
150:       THEN DO;
151:          PRINTER = (39)' ' || COUNTY(RLG.COUNTY_#) ||
152:             '   COMMERCIAL   ' || LNGTH ||
153:             FADT(1,2) || FADT(2,2) || FADT(3,2) || VM(2);
154:          CALL PRINTX (1);
155:          END;
156:       ELSE DO;
157:          PRINTER = (39)' ' || COUNTY(RLG.COUNTY_#) || (16)' ' || LNG
```

```
/* :TRAFFIC-BY-SECTIONS,DATA=XXXXX */

158:          CALL PRINTX (0);
159:          END;

160:     /* OUT OF STATE VEHICLES */
161:     IF ADT(1,3)+ADT(2,3)+ADT(3,3)¬=0 THEN DO;
162:        PRINTER = (57)' ' || 'OUT OF STATE' || (8)' ' ||
163:           FADT(1,3) || FADT(2,3) || FADT(3,3) || VM(3);
164:        CALL PRINTX (1);
165:        END;

166: NEXT_RECORD:
167:     READ FILE (TRAFSUB) SET (PTR_SUB);
168:     GOTO LOOP;

169: DONE:
170:     CLOSE
171:        FILE (TRAFSUB),
172:        FILE (ROADLOG);
173:     CALL EXIT (PARM);

174: END TRBYSEC;
```

SUMMARY-BY-ROUTES --

Member Name	NBT	
Language	PL/I	
Subroutines	PRINTX1	
Files	SYSPRINT	-- IBM messages
	PRINTER	-- SUMMARY-BY-ROUTES output
	TRAFSUB	-- Traffic Summary file
Instruction	1 - 3	"NBT"
	6	Route System ("I," "P," or "S")

SUMMARY-BY-ROUTES prints the summary of rural mileage, rural ADT, and rural vehicle mileage that appears after the traffic-by-sections listing in the Traffic by Sections report. The traffic summary file contains records with the route and system totals; NBT need only read these records, calculate ADT's by dividing the vehicle miles by the route lengths, and print the values.

The NBT program listing follows:

```
/*** TRAFFIC SUMMARY BY ROUTES ***/

 1: /*** TRAFFIC SUMMARY BY ROUTES ***/
 2: NBT: PROCEDURE(PARM) OPTIONS(MAIN);
 3:  DECLARE
 4:   (HEADING(9),PRINTER) CHAR(132) EXT,
 5:   BLANK CHAR(132) INIT(' '),
 6:   PRINTX ENTRY(PIC'Z'),
 7:   PARM CHAR(100),
 8:   INSTR CHAR(80)EXT,
 9:   #_HDGS PIC'Z' DEF INSTR POS (72),
10:   SYSTEM CHAR(1) DEF INSTR POS(6);
11: /* FOR YR OF TRAFFIC FILE */
12:  DECLARE
13:   1 CUR_YR,
14:     2 DUM1 CHAR(1),
15:     2 KEY CHAR(13),
16:     2 DUM2 CHAR(4),
17:     2 YEAR CHAR(2),
18:     2 DUM3 CHAR(76);
19: /* FOR TRAFFIC SUMMARY FILE */
20:  DECLARE
21:   1 SUM,
22:     2 DUM1 CHAR(1),
23:     2 KEY CHAR(13),
24:     2 REMARK CHAR(1),
25:     2 LEN DEC FIXED(7,3),
26:     2 DATA1 CHAR(24),
27:     2 DATA2 CHAR(24),
28:     2 DATA3,
29:       3 (VM,VMFOR,DUM2,VMCOM) DEC FIXED(11,3),
30:    2 DUM3 CHAR(5);
31: /* FOR SYSTEM TOTALS */
32:  DECLARE
33:   1 SYS,
34:     2 DESCR CHAR(11) INIT('  NET TOTAL'),
35:     2 RMI PIC'(7)ZV.Z',
36:     2 AALL PIC'(11)Z',
37:     2 ACOM PIC'(8)Z',
38:     2 AFOR PIC'(8)Z',
39:     2 VMALL PIC'(13)Z',
40:     2 VMCOM PIC'(9)Z',
41:     2 VMFOR PIC'(9)Z',
42:     2 TPALL PIC'(8)ZV.ZZ' INIT(100.00),
43:     2 TPCOM PIC'(5)ZV.ZZ',
44:     2 TPFOR PIC'(5)ZV.ZZ';
45: /* FOR ROUTE TOTALS */
46:  DECLARE
47:   1 RT,
48:     2 DUM1 CHAR(5) INIT(' '),
49:     2 ROUTE_# CHAR(3),
50:     2 RMI PIC'(10)ZV.Z',
51:     2 AALL PIC'(11)Z',
52:     2 ACOM PIC'(8)Z',
53:     2 AFOR PIC'(8)Z',
54:     2 VMALL PIC'(13)Z',
55:     2 VMCOM PIC'(9)Z',
56:     2 VMFOR PIC'(9)Z',
57:     2 PALL PIC'(8)ZV.ZZ',
58:     2 PCOM PIC'(5)ZV.ZZ',
```

```
/*** TRAFFIC SUMMARY BY ROUTES ***/

 59:     2 PFOR PIC'(5)ZV.ZZ';
 60: /* MISC VAR */
 61:  DECLARE
 62:   TRAFSUM FILE RECORD KEYED ENV(INDEXED GENKEY),
 63:   1 TSUM LIKE SUM,
 64:   TMI DEC FIXED(9,3),
 65:   TVM(3) DEC FIXED(9,0),
 66:   KEYT CHAR(13),
 67:   IND BINARY FIXED,
 68:   STRING_RT CHAR(105) DEF RT,
 69:   STRING_SYS CHAR(105) DEF SYS;
 70: /** PGM INITIALIZATION **/
 71: CALL INIT (PARM);
 72:  OPEN FILE(TRAFSUM) INPUT SEQL TITLE('TRAFSUB');
 73:   READ FILE(TRAFSUM) INTO (CUR_YR) KEY('A000');
 74: /* HDGS FOR SUM */
 75:  #_HDGS=7;
 76:    TMI=0;
 77:  HEADING(1)=SUBSTR(BLANK,1,53)||'19'||CUR_YR.YEAR;
 78:  HEADING(2)=SUBSTR(BLANK,1,38)||'INTERSTATE  RURAL  ROU'||
 79:             'TE  SUMMARY';
 80:   IF SYSTEM='P' THEN DO;
 81:     HEADING(2)=SUBSTR(BLANK,1,40)||'PRIMARY  RURAL  ROU'||
 82:                'TE  SUMMARY';
 83:     END;
 84:   IF SYSTEM='S' THEN DO;
 85:     HEADING(2)=SUBSTR(BLANK,1,39)||'SECONDARY  RURAL  R'||
 86:                'OUTE  SUMMARY';
 87:     END;
 88:   HEADING(4)='                                AVERAGE DAILY TRAFFIC '||
 89:     '        *****VEHICLE MILES*****       % OF TOTAL VEH. MILES';
 90:   HEADING(6)='     ROUTE    RURAL          ALL   COMMER- OUT OF'||
 91:     '          ALL    COMMER-  OUT OF        ALL   COMMER- OUT OF';
 92:   HEADING(7)='       NO    MILEAGE    VEHICLES  CIAL    STATE'||
 93:     '      VEHICLES   CIAL     STATE    VEHICLES  CIAL    STATE';
 94: /* TOTALS FOR SYSTEM PRCNT COMPUTATIONS */
 95:  READ FILE(TRAFSUM) INTO(TSUM) KEY(SYSTEM||'999RURAL');
 96: /* CHECK FOR COMMERCIAL & FOREIGN VEH.MILES */
 97: /* ALLOWS FOR CORRECT FINAL PRCNTS */
 98:   IF TSUM.DATA3.VMCOM=0 THEN SYS.TPCOM=0;
 99:     ELSE SYS.TPCOM=100.00;
100:   IF TSUM.DATA3.VMFOR=0 THEN SYS.TPFOR=0;
101:     ELSE SYS.TPFOR=100.00;
102:     READ FILE(TRAFSUM) INTO (SUM) KEY(SYSTEM);
103:   ON ENDFILE(TRAFSUM) GOTO CHECK;
104:   GOTO RO2;
105: RO1:
106:   READ FILE(TRAFSUM) INTO(SUM);
107: /* CK FOR ROUTE TOTAL KEY */
108: RO2:
109:   IF SUM.KEY=SYSTEM||'999RURAL' THEN GOTO SYS_TOT;
110:   IF SUM.KEY¬=SYSTEM||SUBSTR(SUM.KEY,2,3)||'999RURAL' THEN DO;
111:      KEYT=SYSTEM||SUBSTR(SUM.KEY,2,3)||'999RURAL';
112:      READ FILE(TRAFSUM) INTO(SUM) KEY(KEYT);
113:      GOTO RO2;
114:     END;
115: /* ONE ROUTE CALC */
116:   RT.ROUTE_#=SUBSTR(SUM.KEY,2,3);
```

```
/*** TRAFFIC SUMMARY BY ROUTES ***/

117:    RT.RMI=SUM.LEN+.05;
118:    IF SUM.LEN¬=0 THEN DO;
119:    RT.AALL=SUM.DATA3.VM/SUM.LEN+.5;
120:    RT.ACOM=SUM.DATA3.VMCOM/SUM.LEN+.5;
121:    RT.AFOR=SUM.DATA3.VMFOR/SUM.LEN+.5;
122:    END;
123:    ELSE RT.AALL,RT.ACOM,RT.AFOR=0;
124:    RT.VMALL=SUM.DATA3.VM+.5;
125:    RT.VMCOM=SUM.DATA3.VMCOM+.5;
126:    RT.VMFOR=SUM.DATA3.VMFOR+.5;
127:    RT.PALL=(SUM.DATA3.VM/TSUM.DATA3.VM)*100+.005;
128:    IF SUM.DATA3.VMCOM=0 THEN RT.PCOM=0;
129:     ELSE RT.PCOM=(SUM.DATA3.VMCOM/TSUM.DATA3.VMCOM)*100+.005;
130:    IF SUM.DATA3.VMFOR=0 THEN RT.PFOR=0;
131:     ELSE RT.PFOR=(SUM.DATA3.VMFOR/TSUM.DATA3.VMFOR)*100+.005;
132: /* ROUTE PRINT */
133:    PRINTER=STRING_RT;
134:    CALL PRINTX(2);
135:    GOTO RO1;
136: /* SYSTEM CALCS */
137: SYS_TOT:
138:    IND=1;
139:    SYS.RMI=SUM.LEN;
140:    SYS.AALL=TSUM.DATA3.VM/TSUM.LEN+.5;
141:    SYS.ACOM=TSUM.DATA3.VMCOM/TSUM.LEN+.5;
142:    SYS.AFOR=TSUM.DATA3.VMFOR/TSUM.LEN+.5;
143:    SYS.VMALL=TSUM.DATA3.VM;
144:    SYS.VMCOM=TSUM.DATA3.VMCOM;
145:    SYS.VMFOR=TSUM.DATA3.VMFOR;
146: /* SYSTEM TOTAL PRINT */
147:    PRINTER=STRING_SYS;
148:    CALL PRINTX(2);
149: /* CLOSE OF PROG */
150: CHECK:
151:    IF IND¬= 1 THEN GOTO SYS_TOT;
152:    CLOSE FILE(TRAFSUM);
153:    CALL EXIT(PARM);
154:    END NBT;
```